ERSTE AUSGABE · VERSION 1.0

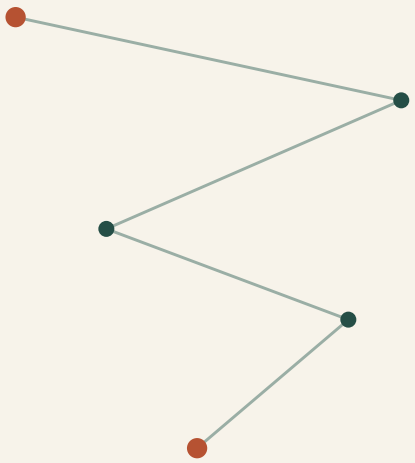
Project Ant Colony

Die andere Evolution der KI

Technische Brüche, verschwundene Randbedingungen und die Frage, was ohne CUDA entstanden wäre

Joachim Müller-Peter

Erste Ausgabe · 30. Juli 2026



Über dieses Buch

Project Ant Colony wird als eigenständiges Buch entwickelt. Es ist weder ein weiterer Teil noch ein Folgeband von *Der Körper der KI*. Beide Werke berühren sich thematisch, setzen aber auf verschiedenen Ebenen an.

Der Körper der KI untersucht KI als gegenwärtige Infrastruktur und Betriebsform. *Project Ant Colony* fragt, wie technische Randbedingungen die historische Evolution möglicher KI-Verfahren geprägt haben - und welche Entwicklungslinien durch andere Selektionsbedingungen sichtbar werden.

Dieses Buch enthält alle 22 Kapitel und das Zwischenspiel seiner Bucharchitektur. Historische Befunde, kontrafaktische Szenarien und die Eröffnung des gegenwärtigen Forschungsprogramms werden sprachlich getrennt. Vollständige Versuchsspezifikationen, Messreihen und technische Ergebnisse gehören in die anschließenden Forschungspapiere.

> **Leitgedanke**
>
> *Algorithmische Verbesserung führt zu bisher übersehenen Möglichkeiten.*

Drei Lesarten

BELEGT	Eine Primärquelle, offizielle Dokumentation oder ein lokales Messartefakt trägt die Aussage direkt.
PLAUSIBEL	Mehrere Anker stützen eine vorsichtige historische oder kontrafaktische Synthese.
PRÜFBAR	Die Synthese wurde in eine heutige, widerlegbare Mess- oder Vergleichsfrage übersetzt.

Inhalt

TEIL I

Wie eine Forschungsfrage entsteht

Der Windkanal öffnet die Frage; Randbedingungen und Bruchfolgen geben ihr ein historisches Instrument.

Kapitel 1 bis 3

KAPITEL 1

Die Abzweigung

Der Windkanal

Der eigene Rechnerverbund erscheint früh als Windkanal; aus seinem Widerstand entsteht die historische Frage.

In einem kleinen Rechnerverbund stehen Maschinen, die für moderne KI-Forschung denkbar ungeeignet wirken. Sie sind alt, ungleich schnell, bandbreitenbegrenzt und nicht für die Lasten gebaut, die heute große Beschleunigercluster füllen.

Gerade deshalb sind sie interessant.

Die Rechner sollen nicht beweisen, dass alte CPUs moderne GPUs ersetzen können. Für viele Operationen wäre dieser Wettkampf physikalisch entschieden. Die Maschinen erfüllen eine andere Aufgabe: Sie bilden einen Windkanal.

Ein Windkanal ist kein Flugzeug. Er übertreibt Widerstände, macht Strömungen sichtbar und zwingt eine Konstruktion, ihre Annahmen offenzulegen. Der Verbund tut dasselbe mit einem Lern- oder Ausführungsverfahren. Was muss gleichzeitig geschehen? Welche Daten müssen sich bewegen? Welche Aufgabe darf warten? Welche Einheit eignet sich unter der aktuellen Last tatsächlich? Wann kostet die Suche nach einer besseren Lösung mehr als die Lösung selbst?

Sobald diese Fragen gestellt werden, verändert sich der Gegenstand. Es geht nicht länger um alte gegen neue Hardware. Es geht darum, ob der Algorithmus als unveränderliche Vorgabe behandelt werden muss - oder ob er auf den gemessenen Zustand einer konkreten Maschine antworten kann.

Im Windkanal wird Geschwindigkeit dabei nur zu einer Messgröße unter mehreren. Ein schneller Lauf kann wertlos sein, wenn er das Qualitätsziel verfehlt. Eine geschickte Verteilung kann mehr Kommunikation verursachen, als sie Rechenzeit spart. Ein guter Plan kann ungültig werden, sobald ein Knoten belastet ist oder ein Messergebnis veraltet. Die widerspenstigen Maschinen machen solche Abhängigkeiten sichtbar, weil ihr Widerstand nicht von reichlich Leistung verdeckt wird.

Damit rückt eine Tätigkeit ins Zentrum, die im fertigen KI-Produkt kaum auffällt: das fortlaufende Abstimmen von Aufgabe, Verfahren und Umgebung. In großen, homogenen Systemen ist viel davon bereits in Bibliotheken und Betriebswissen eingegangen. Im kleinen heterogenen Verbund muss jede Entscheidung wieder begründet werden.

Project Ant Colony beginnt in diesem Windkanal. Um zu verstehen, weshalb seine Frage heute ungewöhnlich erscheint, muss das Buch ihn zunächst verlassen.

Die falsche Frage

Am Anfang stand die Vermutung, ein modernes KI-System könne Verfahren entwerfen, die vorhandene CPUs besser ausnutzen. Sie führte schnell zu der Frage:

> *Können CPUs GPUs ersetzen?*

Das ist die falsche Arena. Ein spezialisierter Beschleuniger besitzt für die Operationen, auf die er ausgelegt wurde, reale Vorteile. Parallelität, Speicherbandbreite, spezialisierte Recheneinheiten und ein ausgereiftes Softwareökosystem lassen sich nicht mit einer eleganten Idee wegdiskutieren.

Hinter dem scheinbaren Hardwarevergleich liegt jedoch eine andere Frage:

> *Welche Formen von KI werden wahrscheinlich, wenn eine bestimmte technische Infrastruktur verfügbar ist - und welche werden unwahrscheinlich, obwohl sie theoretisch möglich bleiben?*

Damit führt die Spur in die Geschichte. Eine technische Öffnung macht nicht nur etwas möglich. Sie verändert, woran anschließend gearbeitet wird. Sie lenkt Rechenzeit, Kapital und Aufmerksamkeit. Werkzeuge, Benchmarks und Ausbildung wachsen um den erfolgreichen Weg. Andere Verfahren werden nicht zwingend widerlegt; sie werden schwerer zu sehen.

Die Entwicklung künstlicher Intelligenz ist daher mehr als eine Folge abstrakter Ideen. Jede Idee benötigt einen Körper aus Maschinen, Speicher, Datenwegen, Energie, Software und menschlichem Wissen. Dieser Körper entscheidet nicht allein. Aber er wählt mit.

Der Highway

CUDA steht in diesem Buch für eine besonders folgenreiche Verbindung. Vor CUDA wurde bereits auf Grafikprozessoren allgemein gerechnet. Doch der Zugang war schwierig. Probleme mussten in Grafikbegriffe übersetzt oder über spezielle Stream-Abstraktionen beschrieben werden.

CUDA verwandelte programmierbare Parallelität schrittweise in einen zugänglichen Entwicklungsweg. Compiler, Bibliotheken, Frameworks, Hardware und Ausbildung verstärkten einander. Aus einer Nebenstraße wurde ein Highway.

Das war realer Fortschritt. Der Highway verkürzte Wege, machte Experimente wiederholbarer und verband Forschung mit einer wachsenden industriellen Infrastruktur. Gerade sein Erfolg hatte jedoch eine zweite Wirkung: Er prägte, welche Modelle leicht zu bauen, zu skalieren und öffentlich zu vergleichen waren.

Ein Highway verändert die Landschaft, durch die er führt. Unternehmen siedeln sich an seinen Ausfahrten an. Karten zeigen ihn deutlicher als Feldwege. Irgendwann wirkt sein Verlauf selbstverständlich.

Was wäre geschehen, wenn dieser Highway nicht entstanden wäre?

Die Frage behauptet weder, KI wäre ohne CUDA stehen geblieben, noch, eine andere Welt wäre automatisch offener oder gerechter geworden. Sie öffnet nur einen kontrollierten Abzweig. CPU-Cluster, offene Standards, Spezialchips und algorithmische Sparsamkeit waren reale Möglichkeiten. Wie sie unter anderem Selektionsdruck zusammengewirkt hätten, bleibt zunächst offen.

Der Auftrag des Buches

Die alternative Geschichte ist kein Selbstzweck. Eine Vergangenheit ohne CUDA lässt sich nicht nachträglich messen. Aber das Gegenbild kann Annahmen der Gegenwart wieder sichtbar machen.

Vielleicht sind manche Nebenwege zu Recht verschwunden. Vielleicht bleiben sie unter bestimmten Bedingungen tragfähig. Beides darf das spätere Experiment zeigen.

Die historische Erzählung erhält dadurch eine Grenze. Sie soll nicht beweisen, dass die Gegenwart falsch abgebogen ist. Sie soll erklären, warum bestimmte Entwurfsentscheidungen heute wie natürliche Eigenschaften von KI erscheinen. Erst wenn diese Selbstverständlichkeiten wieder als Entscheidungen erkennbar sind, können sie fair verglichen werden.

Das Buch folgt deshalb einer Bewegung in drei Richtungen. Es rekonstruiert zunächst historische Öffnungen und ihre Folgen. Dann entfernt es den CUDA-Highway für einen begrenzten Gedankenversuch. Schließlich kehrt es zu den alten Rechnern zurück.

Erst bei dieser Rückkehr wird aus dem Windkanal ein Forschungsinstrument. Dann lautet die Frage nicht mehr, ob eine schwache Maschine stark genug ist. Gefragt wird, ob ein begrenztes und kontrollierbares Entscheidungssystem aus ihrem gemessenen Zustand einen besseren gültigen Ausführungsweg wählen kann als eine einfache feste Regel.

Dabei bleibt ein Vorschlag von seiner Ausführung getrennt. Ein lernendes Modell darf Möglichkeiten erkennen, aber nicht eigenmächtig Sicherheits-, Qualitäts- oder Ressourcengrenzen verändern. Diese Trennung wird später wichtig. An dieser Stelle genügt ihr Zweck: Das Experiment soll die Anpassungsfähigkeit einer Entscheidung prüfen, nicht die Kontrolle über die Maschine aufgeben.

Die Antwort kann Nein lauten.

Damit sie überhaupt etwas bedeutet, muss zuvor verstanden werden, wie technische Bedingungen aus Möglichkeiten dominante Pfade machen. Genau dort beginnt die historische Arbeit.

KAPITEL 2

Randbedingungen sind keine Kulisse

Fünf Kräfte entscheiden mit

Idee, Maschine, Messung, Institution und gesellschaftliche Resonanz wählen gemeinsam, was praktisch sichtbar wird.

Technische Ideen werden gern so erzählt, als träten sie in einem neutralen Wettbewerb gegeneinander an. Jemand formuliert eine Methode, ein Experiment prüft sie, und die bessere Idee setzt sich durch.

In der Welt benötigt jede Idee jedoch Bedingungen, unter denen sie sichtbar werden kann. Ein Lernverfahren braucht eine Maschine, auf der es rechtzeitig zu einem Ergebnis kommt. Die Maschine braucht Werkzeuge und Menschen, die sie beherrschen. Ein Modell braucht Daten und eine Messordnung. Ein Versuch braucht Rechenzeit, Geld und eine Institution, die ihn lange genug trägt.

Mindestens fünf Kräfte wirken dabei zusammen:

1. Die **wissenschaftliche Idee** bestimmt, was formal beschrieben und gelernt werden kann.
2. Die **materielle Technik** bestimmt, was oft genug ausführbar ist, um zum Forschungsprogramm zu werden.
3. **Daten und Messbarkeit** bestimmen, welche Unterschiede sichtbar und vergleichbar werden.
4. **Institutionen und Kapital** bestimmen, welche Ansätze viele Versuche und ein dauerhaftes Ökosystem erhalten.
5. Die **gesellschaftliche Resonanz** bestimmt, welche Versprechen Aufmerksamkeit gewinnen und welche Kosten akzeptiert werden.

Keine dieser Kräfte erklärt eine Entwicklung allein. Ihre Passung erzeugt historische Wirksamkeit.

> *Randbedingungen sind keine Kulisse. Sie gehören zum handelnden Ensemble.*

Passung ist kein Gegenbeweis

Sara Hooker beschreibt mit der *Hardware Lottery* einen Auswahleffekt: Forschungsrichtungen profitieren, wenn ihre Anforderungen zu verfügbarer Hard- und Software passen.[1] Der Gedanke ist wichtig, darf aber nicht zur Entschuldigung jeder unterlegenen Idee werden.

Manche Verfahren verlieren, weil sie schlechter sind. Andere verlieren, weil ihre Vorteile in den verwendeten Aufgaben nicht zählen. Wieder andere setzen eine Hardwareeigenschaft oder ein Werkzeug voraus, das noch nicht existiert. Im Moment der Entscheidung lassen sich diese Fälle oft nicht sauber trennen.

Materielle Begünstigung macht einen wissenschaftlichen Erfolg nicht ungültig. Tiefe neuronale Netze können leistungsfähig sein und zugleich von Daten, Parallelhardware, Bibliotheken und Kapital profitiert haben.

Man kann sich zwei Verfahren vorstellen, die dasselbe Qualitätsziel erreichen. Das erste benötigt viele gleichförmige Rechenoperationen und passt gut zu einem verbreiteten Beschleuniger. Das zweite rechnet weniger, verlangt aber unregelmäßige Speicherzugriffe und komplizierte Koordination. Auf dem Papier kann das zweite sparsamer wirken. In der verfügbaren Werkzeugkette kann es trotzdem teurer, fehleranfälliger und langsamer sein. Sein Nachteil gehört dann weder ausschließlich zum Algorithmus noch ausschließlich zur Maschine. Er entsteht aus ihrer gegenwärtigen Verbindung.

Die relevante Frage lautet deshalb nicht, ob sich die falsche Technik durchgesetzt hat. Sie lautet:

> *Welche Eigenschaften des Siegers stammen aus der Aufgabe - und welche aus der Umgebung, in der der Wettbewerb stattfand?*

Zwischen der Behauptung, jeder Sieger sei zwangsläufig überlegen, und der Behauptung, jeder Sieg sei bloßer Zufall, liegt die Untersuchung.

Wenn Erfolg sich selbst verstärkt

Technische Pfade entwickeln eine Rückkopplung.

Eine Hardware macht bestimmte Operationen günstig. Forschende entwerfen Modelle, die diese Operationen nutzen. Bibliotheken optimieren sie, gute Ergebnisse erhöhen die Nachfrage, und Hersteller verbessern genau die Einheiten, die dem erfolgreichen Modellpfad helfen. Hochschulen lehren den Stack. Unternehmen stellen Menschen ein, die ihn beherrschen.

Jeder Schritt ist vernünftig. Gemeinsam verändern sie den Möglichkeitsraum.

Die Rückkopplung steigert Produktivität, weil nicht jedes Experiment seine Werkzeuge neu erfinden muss. Gleichzeitig verteuert sie Abweichungen. Eine alternative Idee

konkurriert dann nicht nur mit einem anderen Modell, sondern mit Dokumentation, Bibliotheken, Ausbildung, Cloudangeboten, Benchmarks und Beschaffungswegen.

Der erfolgreiche Standard wird Infrastruktur. Seine größte Macht zeigt sich, wenn er aus der Aufmerksamkeit verschwindet.

Was verschwindet, ist nicht nur Hardware. Auch menschliche Arbeit wird unsichtbar: gepflegte Treiber, getestete Bibliotheken, dokumentierte Fehlerbilder, eingespielte Beschaffung und Erfahrung mit dem Betrieb. Ein alternativer Pfad muss diese Leistungen entweder ebenfalls aufbauen oder ihre Abwesenheit als Kosten tragen. Ein fairer Vergleich darf deshalb nicht nur Rechenzeit zählen.

Gesellschaftliche Resonanz ist eine Bruchfolge

Welche Technik sichtbar wird, verändert mehr als die Forschung.

Ein weithin verfügbarer Stack kann Zugang schaffen und zugleich Abhängigkeit von wenigen Anbietern erhöhen. Lokale Ausführung kann Kontrolle über Daten stärken und dennoch ineffizient sein. Längere Hardwarelebensdauer kann Ressourcen sparen und zugleich unsichere Systeme konservieren. Offene Standards können Beteiligung ermöglichen, aber ihre Integrationskosten auf kleine Organisationen abwälzen.

Darum folgt aus einer alternativen technischen Entwicklung nicht automatisch eine gerechtere Gesellschaft. Die Technik verschiebt vielmehr, wer Komplexität beherrschen muss, wer Systeme prüfen kann, wem Daten gehören und welche Kosten sichtbar werden.

Ein zentraler Cloudservice kann den Zugang zu einer komplexen Technik für Millionen Menschen vereinfachen und zugleich Kontrolle konzentrieren. Eine lokale Lösung kann Unabhängigkeit schaffen und zugleich Wartungswissen voraussetzen, das nicht gleich verteilt ist. Offenheit auf der Ebene einer Spezifikation garantiert noch keine praktische Teilhabe. Umgekehrt ist eine integrierte Plattform nicht allein deshalb gesellschaftlich wertlos, weil sie proprietär ist.

Auch Erwartungen wirken zurück. Das Bild einer menschenähnlichen Maschine, das Versprechen wirtschaftlicher Produktivität oder die Angst vor strategischer Abhängigkeit beeinflussen Finanzierung, Regulierung und öffentliche Geduld. Gesellschaft ist nicht nur Empfängerin eines technischen Bruchs. Sie wählt an seinen Folgen mit.

Die Arbeitsregel

Für die folgenden Kapitel gilt daher eine einfache Regel: Eine historische Station wird nicht nach einer großen Ursache beurteilt.

Zuerst ist zu klären, welche Grenze tatsächlich bestand. Dann wird sichtbar, welche Idee, Maschine, Messung und Institution eine Öffnung erzeugten. Erst danach lässt sich fragen, warum ein Weg zum Standard wurde und welche neue Grenze aus seinem Erfolg entstand.

Diese Ordnung schützt vor zwei bequemen Erzählungen. Sie verhindert, dass eine Person, ein Paper oder ein Chip die gesamte Geschichte erklären soll. Und sie verhindert, dass materielle Bedingungen jede wissenschaftliche Leistung in bloße Macht verwandeln.

Was daraus entsteht, ist keine Geschichte ohne Entscheidungen. Es ist eine Geschichte, in der die Bedingungen der Entscheidung sichtbar bleiben.

Das nächste Kapitel gibt dieser Bewegung ihren Namen: Bruch und Bruchfolge.

Quellenhinweise

1 Sara Hooker, „The Hardware Lottery“, *Communications of the ACM* 64, Nr. 12 (2021), 58-65, DOI: 10.1145/3467017.

KAPITEL 3

Brüche und Bruchfolgen

Fortschritt verändert die nächste Frage

Bruch und Bruchfolge bilden ein knappes Instrument, das von nun an an der Geschichte erprobt wird.

Ein wissenschaftlicher Durchbruch wird häufig als Antwort erzählt. Ein Problem war ungelöst, eine Methode verschob die Grenze, anschließend wusste die Welt mehr als zuvor.

Für die Geschichte künstlicher Intelligenz reicht dieses Bild nicht. Ein Durchbruch verändert auch die Fragen, die danach vernünftig erscheinen. Er beeinflusst, welche Experimente bezahlbar sind, welche Messwerte Karriere machen, welche Fähigkeiten gelehrt werden und welche Infrastruktur als selbstverständlich gilt.

Dieses Buch unterscheidet deshalb zwei Bewegungen.

Ein **Bruch** ist der Moment, in dem eine zuvor wirksame Grenze praktisch, theoretisch oder institutionell verschoben wird.

Eine **Bruchfolge** ist die Kette von Veränderungen, die daraus entsteht: neue Methoden, Werkzeuge, Institutionen und Erwartungen - und schließlich eine neue Grenze.

Der Bruch kann in einem Paper, einem Gerät oder einem Wettbewerb sichtbar werden. Seine Folgen verteilen sich über Jahre.

Sechs Glieder

Bruchfolgen lassen sich als wiederkehrende, aber nicht mechanische Bewegung lesen:

1. **Eine Grenze wird spürbar.** Ein Modell lässt sich nicht trainieren, eine Datenmenge nicht verarbeiten oder ein Verfahren nicht wirtschaftlich betreiben.
2. **Eine Öffnung entsteht.** Eine Methode, ein Gerät, ein Datensatz, eine Schnittstelle oder eine Institution macht etwas praktisch.
3. **Ein Auswahlvorteil wird sichtbar.** Die neue Kombination liefert bessere, schnellere oder leichter wiederholbare Ergebnisse.
4. **Ein Ökosystem verstärkt den Erfolg.** Bibliotheken, Benchmarks, Ausbildung, Produkte und Kapital wachsen um den neuen Pfad.

5. **Die Lösung wird zur stillen Voraussetzung.** Spätere Arbeiten optimieren innerhalb des entstandenen Systems und erwähnen dessen Bedingungen immer seltener.
6. **Aus dem Erfolg entsteht die nächste Grenze.** Rechenbedarf, Kommunikation, Energie, Konzentration oder Integrationsaufwand werden zum neuen Engpass.

Nicht jede Öffnung durchläuft alle sechs Glieder. Manche Ideen bleiben jahrzehntelang verfügbar, bevor ein anderer Teil des Systems zu ihnen passt. Manche Standards gewinnen durch Marktposition, andere durch wissenschaftliche Leistung, viele durch beides. Die Folge ist ein Instrument, kein Naturgesetz.

Eine kleine technische Verbesserung kann diese Bewegung auslösen. Ein Compiler verkürzt die Zeit bis zum ersten brauchbaren Experiment. Dadurch werden mehr Varianten erprobt. Eine davon gewinnt einen Wettbewerb, wird in eine Bibliothek aufgenommen und prägt den nächsten Datensatz. Wenige Jahre später beschreiben neue Arbeiten nur noch Modell und Ergebnis; der Compiler, der den Suchraum einst geöffnet hat, erscheint nicht mehr im Argument.

Die nächste Generation begegnet dem erfolgreichen Verfahren deshalb unter anderen Bedingungen als seine Erfinder. Für sie ist die frühere Öffnung bereits Infrastruktur. Sie beginnt ihre Arbeit hinter einer Grenze, die für die vorige Generation noch sichtbar war. Genau dieser Zeitversatz macht eine Bruchfolge historisch interessant.

Kein Paradigmenwechsel ohne Rest

Das Wort Paradigmenwechsel klingt nach einer sauberen Trennung: vorher falsch, nachher richtig. Tatsächliche Forschungswelten sind unordentlicher. Symbolische Verfahren verschwanden nicht, als neuronale Netze zurückkehrten. CPUs hörten nicht auf zu rechnen, als GPUs das Training großer Modelle prägten.

Ein Paradigmenwechsel bezeichnet hier deshalb keine vollständige Auslöschung. Er liegt vor, wenn sich die praktisch dominante Verbindung aus Forschungsfrage, Methode, Messung und Infrastruktur verschiebt.

Entscheidend ist der neue Default. Wofür fließt Geld? Welche Ergebnisse gelten als wichtig? Welches Wissen wird vorausgesetzt? Welche Abhängigkeiten müssen nicht mehr genannt werden?

So verstanden ist ein Paradigma nicht nur eine Theorie. Es ist eine Arbeitsordnung.

Diese Arbeitsordnung wirkt auch auf Probleme zurück. Was sich leicht messen und veröffentlichen lässt, wird häufiger untersucht. Was besondere Geräte, lange Anläufe oder ungewöhnliche Qualitätsmaßstäbe verlangt, erhält weniger Versuche. Der Paradigmenwechsel verändert daher nicht nur die bevorzugte Antwort. Er verändert, welche Fragen überhaupt als aussichtsreich gelten.

Die soziale Lebensdauer eines Bruchs

Technische und gesellschaftliche Folgen laufen nicht im selben Takt.

Ein mathematisches Ergebnis kann lange gültig bleiben, während seine institutionelle Bedeutung sich mehrfach verändert. Eine Kritik kann eine ganze Forschungsrichtung prägen, obwohl sie nur eine begrenzte Modellklasse betrifft. Ein spektakulärer Wettbewerbserfolg kann zum Symbol für eine viel größere technische Entwicklung werden.

Die später erzählte Geschichte verdichtet diese Prozesse gern zu einer Heldenfolge. Eine Person erfindet, eine zweite widerlegt, eine dritte rettet. Das ist merkfähig, verschiebt aber die Aufmerksamkeit von den Bedingungen, in denen eine Arbeit wirksam wurde.

Rückblickend entsteht noch eine zweite Täuschung. Weil ein späterer Weg erfolgreich war, erscheinen alle früheren Stationen wie notwendige Vorbereitungen auf dieses Ziel. Verlorene Alternativen werden zu Irrtümern, Übergänge zu zwangsläufigen Stufen. Die Bruchfolge soll diese Ordnung nicht umkehren und aus Verlierern heimliche Sieger machen. Sie hält lediglich offen, wo Auswahl statt Notwendigkeit wirkte.

Die kommenden Kapitel betrachten deshalb McCulloch und Pitts, Hebb, Rosenblatt, Minsky und Papert, Backpropagation, ImageNet, AlexNet und CUDA nicht als Stufen einer vorgezeichneten Treppe. Jede Station verschiebt eine bestimmte Grenze. Jede trifft auf einen anderen materiellen und gesellschaftlichen Körper. Und jede hinterlässt Voraussetzungen, die spätere Forschung leicht mit Naturgesetzen verwechselt.

Die Methode für den weiteren Weg

Für jede Station werden vier Fragen gestellt:

1. Welche Grenze wurde tatsächlich verschoben?
2. Welche Bedingungen machten die Öffnung praktisch wirksam?
3. Welche Bruchfolge entstand aus ihrem Erfolg?
4. Welche neue Grenze oder unsichtbare Voraussetzung blieb zurück?

Diese Fragen erlauben historische Genauigkeit, ohne die Erzählung bei jeder Station erneut abzusichern. Prioritätsfragen, parallele Entwicklungen und institutionelle Wirkungen werden dort behandelt, wo sie konkret werden.

Auch die gesellschaftliche Reflexion folgt den Brüchen. Sie erscheint nicht als nachträglicher Kommentar, sondern dort, wo Technik Zugang, Macht, Erwartungen oder Verantwortung neu verteilt.

Damit ist das Instrument vollständig. Es muss nun nicht weiter erklärt, sondern angewendet werden.

Der nächste Teil beginnt mit einer Öffnung, die unscheinbar und folgenreich zugleich war: Ein Neuron wurde so weit abstrahiert, dass es Gegenstand einer technischen Berechnung werden konnte.

TEIL II

Die Entwicklungslinie, die zum Standard wurde

Vom formalen Neuron über Lernen und Backpropagation bis zur Industrialisierung des Beschleunigerparadigmas.

Kapitel 4 bis 10

KAPITEL 4

Das Neuron wird formalisierbar

Ein Körper wird zu einem Kalkül

Die formale Reduktion des Neurons trifft auf Asimovs kulturelle Frage, unter welchen Regeln Maschinen handeln dürfen.

1943 veröffentlichten Warren S. McCulloch und Walter Pitts einen Text, dessen Titel bereits das Programm verriet: *A Logical Calculus of the Ideas Immanent in Nervous Activity*.[1]

Das Gehirn erschien darin nicht als geheimnisvolle Einheit, sondern als Gegenstand formaler Beschreibung. Neuronen wurden radikal vereinfacht. Sie waren entweder aktiv oder nicht aktiv. Eingehende Signale wurden nach festen Regeln zusammengeführt. Ein Schwellenwert entschied, ob eine Einheit im nächsten Zeitschritt feuerte. Hemmende Signale konnten Aktivität unterbinden.

Aus Sicht heutiger Neurowissenschaft war dies kein realistisches Gehirnmodell. Genau diese Abstraktion war jedoch seine Kraft.

McCulloch und Pitts zeigten, dass sich Netze solcher idealisierten Einheiten mit Aussagenlogik in Beziehung setzen lassen. Bestimmte neuronale Verschaltungen konnten logische Ausdrücke realisieren; rekurrente Verschaltungen eröffneten zeitabhängige Prozesse. Damit entstand eine Brücke zwischen Neurophysiologie, Logik und Berechnung.

> *Das Neuron wurde nicht kopiert. Es wurde übersetzt.*

Diese Übersetzung ist der erste große Bruch der Entwicklungslinie dieses Buches. Ein biologisches Phänomen wurde so stark reduziert, dass es als technische Operation behandelbar wurde.

Die produktive Gewalt der Abstraktion

Jede Abstraktion lässt Eigenschaften zurück.

Reale Nervenzellen besitzen komplexe elektrochemische Dynamiken. Sie verändern sich, arbeiten mit unterschiedlichen Zeitmaßen, reagieren auf Modulatoren und sind in Körper eingebettet. Das McCulloch-Pitts-Neuron kannte davon fast nichts. Es besaß

keinen Stoffwechsel, keine Entwicklungsgeschichte, keine Plastizität und zunächst keinen Lernalgorithmus.

Man könnte dies als Schwäche lesen. Für die frühe Rechenidee war es eine Voraussetzung. Ein Modell, das jede biologische Einzelheit bewahren wollte, wäre kaum mathematisch handhabbar gewesen. Erst das Weglassen machte eine allgemeine Frage formulierbar:

> *Welche geistig erscheinenden Operationen lassen sich als Zusammensetzung einfacher, lokaler Entscheidungen beschreiben?*

Diese Frage reicht bis in die Gegenwart. Ein künstliches Neuron ist heute anders definiert, numerisch kontinuierlich und in großen Tensorprogrammen organisiert. Doch die grundlegende Geste bleibt erkennbar: Verhalten wird in kleine Operationen zerlegt, deren Zusammenspiel eine komplexere Funktion bildet.

Der historische Fehler läge darin, aus dieser Ähnlichkeit eine direkte Linie zum modernen Deep Learning zu zeichnen. McCulloch und Pitts beschrieben kein lernendes Netz im heutigen Sinn. Ihre Arbeit lieferte eine formale Möglichkeitsbedingung, nicht den fertigen Trainingsweg.

Zwischen Nerv und Maschine

Die Arbeit entstand in einem intellektuellen Raum, in dem die Grenzen heutiger Fächer noch nicht festgezogen waren. Logik, Psychiatrie, Neurophysiologie, Elektrotechnik und die entstehende Kybernetik berührten sich. Die Frage, ob Denken berechenbar sein könnte, war weder rein philosophisch noch bereits ein gewöhnliches Ingenieurproblem.

Gerade diese Zwischenlage war folgenreich.

Wer das Nervensystem als logisches Netz beschreibt, verändert beide Seiten des Vergleichs. Der Nerv wird technisch lesbar. Die Maschine erhält eine Sprache, in der sie als Modell von Erkenntnis erscheinen kann. Aus einer Analogie wird ein Forschungsprogramm.

Dabei entstand ein Muster, das die weitere KI-Geschichte begleiten sollte:

1. Ein biologischer oder kognitiver Vorgang wird formal reduziert.
2. Die Reduktion macht eine technische Implementierung möglich.
3. Die technische Implementierung erzeugt eigene Erfolge.
4. Diese Erfolge werden rückwirkend als Erklärung des ursprünglichen Vorgangs gelesen.

Der vierte Schritt ist der riskante. Ein Modell kann eine Aufgabe lösen, ohne den biologischen Mechanismus abzubilden. Technische Funktion und naturwissenschaftliche Erklärung sind nicht identisch.

Asimovs Maschine unter Regeln

Fast zeitgleich entstand in der Science-Fiction ein anderes Bild der denkenden Maschine. Isaac Asimovs Erzählung *Runaround* von 1942 formulierte die drei Robotergesetze ausdrücklich; 1950 erreichten sie mit der Sammlung *I, Robot* ein breiteres Publikum.[2]

Asimov fragte nicht, wie ein Nerv formalisiert oder wie eine Maschine lernen könnte. Seine Geschichten begannen einen Schritt später: Wenn Maschinen handeln, wie werden ihre Befugnisse begrenzt? Was geschieht, wenn einfache Regeln miteinander in Konflikt geraten?

Eine direkte Wirkung auf McCulloch und Pitts ist damit nicht belegt und für diesen Zusammenhang auch nicht nötig. Wissenschaft und Science-Fiction antworteten auf einen gemeinsamen kulturellen Horizont. Die eine Seite machte geistig erscheinende Vorgänge berechenbar. Die andere stellte sich bereits vor, welche gesellschaftlichen Verträge solche Maschinen benötigen würden.

Für Project Ant Colony laufen beide Fragen wieder zusammen. Ein Modell kann Möglichkeiten erzeugen. Ob es handeln darf, ist eine andere Entscheidung.

Project Ant Colony übernimmt deshalb die formale Kühnheit der frühen Arbeit, nicht ihre mögliche Überdehnung. Abstraktionen sind Werkzeuge. Ihr Erfolg gibt ihnen keinen Anspruch, alles Verlassene für unwichtig zu erklären.

Eine Grenze verschwindet

Vor McCulloch und Pitts war die Vorstellung mechanisierbarer Nervenaktivität keineswegs unbekannt. Neu war die Präzision, mit der neuronale Ereignisse und logische Operationen in einen gemeinsamen Kalkül gebracht wurden. Eine unscharfe Analogie wurde zu etwas, das man beweisen, kombinieren und widerlegen konnte.

Damit verschwand eine Grenze: die Grenze zwischen der Behauptung, ein Nervensystem *verhalte sich irgendwie wie* eine Maschine, und der Möglichkeit, eine konkrete abstrakte Maschine aus neuronalen Einheiten zu beschreiben.

Die Bruchfolge war größer als das Modell selbst.

- Neuronale Aktivität wurde zum Gegenstand formaler Berechnung.
- Netzwerke einfacher Einheiten konnten als Träger komplexer Funktionen gedacht werden.
- Biologische Inspiration und technische Realisierung begannen sich zu trennen.
- Die Frage nach maschineller Intelligenz erhielt eine neue elementare Baueinheit.

Zugleich entstand sofort eine neue Grenze. Ein fest verdrahtetes logisches Netz kann eine Funktion ausführen. Es erklärt noch nicht, wie die richtige Verschaltung entsteht.

Form ist noch kein Lernen

In heutigen Darstellungen werden McCulloch und Pitts häufig an den Beginn der Geschichte neuronaler Netze gestellt. Das ist sinnvoll, solange der Abstand zum Lernen sichtbar bleibt.

Ein Netz kann formal mächtig sein und praktisch unbrauchbar, wenn jede Verbindung von Hand festgelegt werden muss. Für kleine logische Schaltungen ist dies denkbar. Für Wahrnehmung, Sprache oder Anpassung an eine wechselnde Umwelt wird es zur zentralen Grenze.

Die nächste Entwicklungsstufe musste deshalb nicht nur fragen, was ein Netz darstellen kann. Sie musste fragen, wie seine Struktur oder seine Gewichte aus Erfahrung verändert werden.

Diese Verschiebung ist tiefgreifend. Beim Programmieren wird der gewünschte Vorgang in Regeln übersetzt. Beim Lernen wird ein Verfahren definiert, das aus Beispielen oder Rückmeldungen eine interne Ordnung erzeugen soll.

Das Problem wandert damit:

> *Nicht mehr allein: Welche Schaltung löst die Aufgabe? Sondern: Welche lokale Veränderung lässt eine geeignete Schaltung entstehen?*

McCulloch und Pitts hatten das Neuron formalisierbar gemacht. Hebb und Rosenblatt würden das veränderbare Netz ins Zentrum rücken.

Die erste Lektion für Ant Colony

Der historische Abstand von 1943 zur Gegenwart ist groß. Die methodische Lektion ist unmittelbar.

Project Ant Colony muss seine Maschine ebenfalls abstrahieren. Ein reales Cluster besitzt unzählige Zustände: Prozesse, Temperaturen, Cacheeffekte, Netzlast, Gastmaschinen, Speicherwege und Fehler. Ein Entscheidungsmodell kann nicht jede Einzelheit in voller Auflösung behandeln.

Es benötigt eine formale Einheit - keinen künstlichen Nerv, sondern einen Zustandsvektor.

Die Qualität dieser Abstraktion entscheidet mit über das Ergebnis. Wird zu viel weggelassen, plant das System auf einer fiktiven Maschine. Wird zu viel aufgenommen, wird der Zustandsraum unbeherrschbar und jede Entscheidung veraltet, bevor sie ausgeführt ist.

McCulloch und Pitts erinnern daran, dass produktive Forschung häufig mit einer gewagten Reduktion beginnt. Sie erinnern ebenso daran, diese Reduktion nicht mit der Welt zu verwechseln.

Quellenhinweise

1 Warren S. McCulloch und Walter Pitts, „A Logical Calculus of the Ideas Immanent in Nervous Activity“, *The Bulletin of Mathematical Biophysics* 5 (1943), 115-133, DOI: 10.1007/BF02478259.

2 American Museum of Natural History, „Revisiting Asimov's Three Laws of Robotics“, zur Veröffentlichung von *Runaround* 1942 und *I, Robot* 1950.

KAPITEL 5

Lernen wird zu einer Maschinenoperation

Von der Schaltung zur Veränderung

Hebb und Rosenblatt verschieben die Maschine vom festen Programm zum Lernen - im Zeitalter von Sputnik und Perry Rhodan.

Ein festes Netz kann komplexe Funktionen ausführen. Intelligenz verlangt jedoch mehr als eine gelungene Verdrahtung. Sie verlangt Anpassung.

Donald O. Hebb formulierte 1949 in *The Organization of Behavior* eine neuropsychologische Theorie, in der die Veränderung von Verbindungen eine zentrale Rolle spielte.[1] Wenn die Aktivität einer Zelle wiederholt zur Aktivierung einer anderen beiträgt, soll die Wirksamkeit ihrer Verbindung zunehmen. Hebb verband diese Annahme mit *cell assemblies* und zeitlichen Folgen solcher Verbände.

Der bekannte Satz „cells that fire together wire together“ stammt nicht wörtlich aus Hebbs Buch. Als Merksatz trifft er einen Teil der Idee, lässt aber ihren theoretischen Rahmen verschwinden.

Für die Technikgeschichte war vor allem eine Verschiebung wichtig: Verbindungsstärken mussten nicht für immer feststehen. Lernen konnte als lokale Änderung eines Netzes gedacht werden.

Das war noch kein moderner Optimierungsalgorithmus. Es war ein Prinzip, das Lernen von einer äußeren Programmierhandlung in die Maschine verlegte.

Die Verheißung des Lokalen

Lokale Lernregeln besitzen eine besondere Anziehungskraft. Eine Einheit muss nicht das gesamte System verstehen. Sie verändert sich aufgrund von Signalen, die an ihrer Verbindung verfügbar sind. Aus vielen begrenzten Anpassungen kann eine globale Ordnung entstehen.

Diese Idee berührt bereits das spätere Ameisenmodell. Eine Ameise kennt nicht den vollständigen Plan der Kolonie. Ein Worker kennt nicht den vollständigen Zustand eines Clusters. Lokale Regeln können dennoch ein brauchbares Gesamtverhalten erzeugen.

Die Analogie darf nicht überdehnt werden. Hebbs Theorie ist weder ein Cluster-Scheduler noch Ant-Colony-Optimization. Ihr Wert liegt in einer grundsätzlichen Operation:

> *Eine Struktur wird nicht ausschließlich entworfen. Sie kann durch Erfahrung geformt werden.*

Damit entstand eine neue Erwartung an Maschinen. Sie sollten nicht nur Anweisungen abarbeiten, sondern ihre interne Reaktion aus Beispielen verändern.

Rosenblatts Schritt

Frank Rosenblatt gab dieser Erwartung Ende der 1950er Jahre eine mathematische, experimentelle und öffentliche Form. Sein 1958 veröffentlichter Text beschrieb das Perzeptron als probabilistisches Modell für Informationsspeicherung und Organisation im Gehirn.[2]

Rosenblatts Perzeptron war nicht nur eine einzelne Gleichung. Es war ein Forschungsprogramm aus Theorie, Lernregeln und Maschinenentwürfen. Eingaben wurden über anpassbare Gewichte mit einer Ausgabeeinheit verbunden. Aus Beispielen konnte das System seine Entscheidung verändern.

Im einfachsten heute gelehrten Fall bildet das Perzeptron eine lineare Entscheidungsgrenze. Für korrekt trennbare Daten kann ein Aktualisierungsverfahren eine passende Gewichtskonfiguration finden. Der entscheidende kulturelle Effekt lag jedoch weniger in der Geometrie als in der sichtbaren Operation:

Eine Maschine wurde nicht für jedes Muster einzeln programmiert. Sie wurde trainiert.

Ein neues Verhältnis zum Fehler

Lernen verändert die Rolle des Fehlers.

In einem klassischen Programm ist ein falsches Ergebnis häufig ein Defekt der Regel oder ihrer Implementierung. In einem lernenden System wird die Abweichung zum Signal. Das System vergleicht eine Ausgabe mit einer gewünschten Reaktion und verändert seine Parameter.

Damit entstehen neue Fragen:

- Welche Beispiele repräsentieren die Aufgabe?
- In welcher Reihenfolge werden sie gezeigt?
- Welche Fehler führen zu welcher Veränderung?
- Wann gilt das Lernen als abgeschlossen?
- Was geschieht bei widersprüchlichen oder nicht trennbaren Daten?
- Generalisiert die gefundene Grenze auf neue Fälle?

Die Maschine gewinnt Anpassungsfähigkeit. Der Mensch verliert dafür einen Teil der direkten Kontrolle über die entstehende interne Ordnung.

Diese Verschiebung ist bis heute nicht abgeschlossen. Moderne Modelle besitzen Milliarden Parameter, doch die erkenntnistheoretische Spannung ist dieselbe: Wir bestimmen Verfahren, Daten und Zielgröße; die konkrete interne Lösung entsteht im Training.

Theorie, Maschine und Öffentlichkeit

Das Perzeptron wurde zu einem frühen Beispiel dafür, dass eine technische Idee gleichzeitig in mehreren Arenen lebt.

In der wissenschaftlichen Arena ging es um Darstellungsfähigkeit, Konvergenz und biologische Plausibilität. In der technischen Arena ging es um Sensoren, Schaltungen und veränderbare Gewichte. In der institutionellen Arena ging es um Förderung und die Aussicht auf lernende Erkennungssysteme. In der Öffentlichkeit wurde daraus rasch die Vorstellung einer Maschine, die sehen, erkennen und vielleicht eines Tages denken könne.

Mikel Olazarans wissenschaftssoziologische Rekonstruktion zeigt, dass die spätere Perzeptron-Kontroverse nicht nur aus neutral aufeinanderfolgenden Beweisen bestand. Unterschiedliche Forschungsgruppen deuteten Reichweite, Grenzen und Zukunft neuronaler Verfahren verschieden; Autorität und Ressourcenverteilung gehörten zur Geschichte.[3]

Das macht Rosenblatts Arbeit weder richtiger noch falscher. Es erklärt, warum ihre Bruchfolge größer war als ihr mathematischer Kern.

Zwischen Sputnik und Perry Rhodan

Rosenblatts Perzeptron erschien in einer Zeit, in der technische Zukunft öffentlich beinahe körperlich spürbar wurde. Am 4. Oktober 1957 brachte die Sowjetunion mit Sputnik den ersten künstlichen Satelliten in eine Erdumlaufbahn. Der kleine Sender eröffnete nicht nur das Raumfahrtzeitalter. Er verschärfte den technologischen Wettbewerb des Kalten Krieges und trug zur Gründung der NASA im folgenden Jahr bei.[4]

In diesem Klima war die lernende Maschine mehr als ein Laborexperiment. Sie passte in eine Welt, die Technik als geopolitische Kraft, Zukunftsversprechen und öffentlichen Wettkampf zugleich erlebte.

Auch die populäre Vorstellungskraft erhielt einen eigenen Takt. Am

8. September 1961 erschien im deutschsprachigen Raum das erste

Perry-Rhodan-Heft. Seine Handlung schickte einen amerikanischen Astronauten auf eine damals noch zukünftige Mondmission und von dort in eine galaktische Geschichte.[5] Woche für Woche wurde technische Zukunft damit zu Alltagslektüre.

Rosenblatt, Sputnik und Perry Rhodan bilden keine gemeinsame Ursache. Sie zeigen drei verschiedene Ebenen derselben Epoche: eine wissenschaftliche Idee, eine technische Machtdemonstration und ihr kulturelles Echo. Lernen, Raumfahrt und intelligente Maschinen waren nicht dasselbe. Doch sie teilten die Erwartung, dass scheinbar feste Grenzen bald verschoben werden könnten.

Die Grenze der sichtbaren Klasse

Der Erfolg des Perzeptrons schuf eine neue Grenze. Sobald Lernen als Maschinenoperation sichtbar war, musste gefragt werden, welche Probleme die gewählte Architektur überhaupt darstellen konnte.

Ein einfaches Perzeptron kann nur Aufgaben lösen, deren Klassen sich durch eine geeignete lineare Grenze trennen lassen. Probleme mit anderen Abhängigkeiten verlangen zusätzliche Struktur, andere Merkmale oder mehrere Verarbeitungsschichten.

Diese Grenze war nicht bloß ein nachträglicher Einwand. Sie gehört zum wissenschaftlichen Kern. Ein Lernverfahren kann nur innerhalb des Darstellungsraums seiner Architektur suchen. Wenn die richtige Funktion dort nicht existiert, helfen mehr Daten und längeres Training nicht.

Die Unterscheidung zwischen **Lernen** und **Darstellen** ist für das gesamte Buch grundlegend:

- Die Architektur bestimmt, welche Lösungen möglich sind.
- Das Lernverfahren bestimmt, welche davon gefunden werden.
- Die Hardware bestimmt mit, welche Suche praktisch ausführbar ist.
- Die Messordnung bestimmt, welche gefundene Lösung als Erfolg gilt.

Diese vier Ebenen werden später im CUDA-Zeitalter eng zusammenrücken.

Was das frühe Lernen noch nicht konnte

Hebb und Rosenblatt entfernten eine Grenze, aber nicht alle.

Hebbs lokale Plastizitätsannahme erklärte nicht automatisch, wie eine Maschine komplexe überwachte Aufgaben zuverlässig löst. Das Perzeptron besaß eine konkretere Anpassungsregel, blieb in seiner einfachen Form aber auf eine begrenzte Funktionsklasse beschränkt. Mehrschichtige Netze waren als Idee vorstellbar, doch die gezielte Zuordnung eines Gesamtfehlers zu verborgenen Verbindungen war praktisch und konzeptionell schwierig.

Die neue Grenze lautete:

> *Wie lernt eine interne Einheit, deren gewünschte Ausgabe niemand direkt vorgibt?*

Eine Ausgabeeinheit kann mit einem bekannten Ziel verglichen werden. Eine verborgene Einheit liegt zwischen Eingabe und Ausgabe. Ihr Beitrag zum Fehler ist vermittelt. Ohne Verfahren zur Zuweisung dieses Beitrags bleibt Tiefe eine architektonische Möglichkeit ohne robusten Trainingsweg.

Die zweite Lektion für Ant Colony

Auch Project Ant Colony unterscheidet zwischen einem erlaubten Entscheidungsraum und einem Verfahren, das darin sucht.

Ein trainiertes MOS-Modell kann keine gute Strategie wählen, wenn die brauchbare Strategie im Aktionsschema nicht ausdrückbar ist. Umgekehrt genügt ein großer Aktionsraum nicht, wenn das Modell aus Messwerten keine verlässlichen Entscheidungen lernt.

Die frühe Geschichte des Perzeptrons warnt vor zwei Kurzschlüssen:

1. Aus der Fähigkeit zu lernen folgt keine unbegrenzte Darstellungsfähigkeit.
2. Aus einer theoretisch darstellbaren Lösung folgt nicht, dass ein Lernverfahren sie unter realen Bedingungen findet.

Project Ant Colony wird daher jede Entscheidungsebene getrennt prüfen: Was ist erlaubt? Was ist darstellbar? Was wird vorgeschlagen? Was besteht die deterministische Prüfung? Was verbessert am Ende eine gemessene Zielgröße?

Lernen wird erst dann zur belastbaren Maschinenoperation, wenn diese Grenzen sichtbar bleiben.

Quellenhinweise

1 Donald O. Hebb, *The Organization of Behavior: A Neuropsychological Theory*, New York: John Wiley and Sons, 1949.

2 Frank Rosenblatt, „The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain“, *Psychological Review* 65, Nr. 6 (1958), 386-408, DOI: 10.1037/h0042519.

3 Mikel Olazaran, „A Sociological Study of the Official History of the Perceptrons Controversy“, *Social Studies of Science* 26, Nr. 3 (1996), 611-659, DOI: 10.1177/030631296026003005.

4 NASA History, „Sputnik and the Dawn of the Space Age“, zum Start von Sputnik 1 am 4. Oktober 1957 und seiner institutionellen Wirkung.

5 Westdeutscher Rundfunk, „8. September 1961 - Erstes Perry-Rhodan-Heft erscheint“, zeitgeschichtlicher Rückblick.

KAPITEL 6

Eine Grenze wird sichtbar

Das zu saubere Märchen

Minsky und Papert formulieren Grenzen in einem Jahr zwischen Mondlandung, Vietnamprotest, Woodstock und Ernüchterung.

Die bekannteste Kurzgeschichte der neuronalen Netze besitzt drei Akte.

Frank Rosenblatt baut das Perzeptron. Marvin Minsky und Seymour Papert beweisen 1969, dass es entscheidende Aufgaben nicht lösen kann. Die Forschung verliert das Interesse, bis Backpropagation in den 1980er Jahren neuronale Netze wiederbelebt.

Die Erzählung ist nicht vollständig falsch. Sie ist gerade so richtig, dass sie eine kompliziertere Geschichte verdecken kann.

Minsky und Papert veröffentlichten mit *Perceptrons: An Introduction to Computational Geometry* eine mathematische Untersuchung abstrakter Perzeptrons.[1] Sie präzisierten reale Grenzen bestimmter Klassen. Das Buch war ein wissenschaftliches Ereignis und wurde als Urteil über die Zukunft neuronaler Verfahren gelesen.

Aber ein Buch verursacht nicht allein die institutionelle Bewegung eines ganzen Feldes. Finanzierungsentscheidungen, überzogene Erwartungen, technische Schwierigkeiten, konkurrierende Programme und bereits vorhandene Machtstrukturen wirkten zusammen.

Was kritisiert wurde

Ein einfaches Perzeptron berechnet eine gewichtete Summe seiner Eingaben und entscheidet anhand eines Schwellenwerts. Geometrisch trennt es den Eingaberaum durch eine Hyperebene.

Damit sind manche Aufgaben leicht darstellbar und andere grundsätzlich nicht. Das häufig verwendete Lehrbeispiel ist XOR: Keine einzelne lineare Grenze trennt die beiden positiven Fälle von den beiden negativen. Minsky und Paperts Arbeit war breiter als dieses Beispiel. Sie untersuchten Eigenschaften bestimmter lokaler und linearer Perzeptronklassen und behandelten unter anderem Aufgaben wie Parität und Zusammenhang.

Die Kritik traf einen wunden Punkt. Die öffentliche und institutionelle Verheißung des Perzeptrons war größer geworden als die gesicherte Leistungsfähigkeit einfacher Architekturen.

Eine mathematisch bewiesene Grenze ist kein Rückschritt. Sie verhindert, dass mehr Geld und mehr Daten in einen Darstellungsraum fließen, der die gesuchte Lösung nicht enthält.

Der Bruch von 1969 war deshalb zunächst eine Präzisierung.

Was nicht bewiesen wurde

Aus den Grenzen einer Schicht folgt nicht die Unmöglichkeit aller mehrschichtigen neuronalen Netze. Zusätzliche Schichten können nichtlineare Zwischenrepräsentationen bilden und dadurch Funktionen darstellen, die einer einzelnen linearen Einheit verschlossen bleiben.

Das Problem lag darin, dass „mehr Schichten“ noch kein praktikables Forschungsprogramm ergab. Für verborgene Einheiten fehlten weithin akzeptierte und gut funktionierende Trainingsverfahren. Rechenleistung war begrenzt. Datensätze und Werkzeuge waren klein. Biologische Plausibilität, technische Umsetzbarkeit und mathematische Analyse zogen nicht in dieselbe Richtung.

Minsky und Papert mussten also nicht beweisen, dass jede denkbare Tiefe unmöglich war, damit ihre Kritik institutionell weit wirkte. Es genügte, dass die naheliegende und finanzierte Form des Perzeptrons an klaren Grenzen scheiterte, während der alternative Weg unsicher und teuer blieb.

Die historische Wirkung einer Grenze ist nicht identisch mit ihrem logischen Aussageumfang.

Die „offizielle Geschichte“

Olazaran rekonstruierte 1996, wie aus der Perzeptron-Kontroverse später eine offizielle Geschichte wurde.[2] In dieser Erinnerung erschien die Kritik als eindeutiger wissenschaftlicher Abschluss eines fehlgeleiteten Programms. Die Wiederkehr neuronaler Netze ließ dieselbe Episode anschließend als tragische Unterbrechung eines eigentlich richtigen Weges erscheinen.

Beide Lesarten ordnen die Vergangenheit aus Sicht des späteren Siegers.

Die erste legitimiert den Aufstieg symbolischer KI: Neuronale Ansätze waren begrenzt, also war die Verlagerung von Ressourcen vernünftig.

Die zweite legitimiert den späteren Connectionismus: Neuronale Netze waren grundsätzlich richtig, wurden aber durch eine zu weit interpretierte Kritik aufgehalten.

Die historische Wirklichkeit war weniger symmetrisch. Forschende verstanden die Ergebnisse unterschiedlich. Neuronale Arbeit verschwand nicht überall. Symbolische Systeme erzielten eigene Erfolge und besaßen gute Gründe für ihre Methoden. Institutionen suchten Programme, die auf den verfügbaren Computern und für die geförderten Aufgaben Fortschritt versprachen.

Ein Paradigma gewinnt selten nur, weil der Gegner einen Fehler macht.

1969: Mond, Krieg und Ernüchterung

Das Erscheinungsjahr von *Perceptrons* trägt eine eigentümliche Spannung.

Am 20. Juli 1969 landeten mit Apollo 11 erstmals Menschen auf dem Mond. Wenige Wochen später wurde Woodstock zum kulturellen Bild einer anderen Aufbruchserzählung. Zugleich war das Jahr von sozialen Konflikten, Rassismus, dem Vietnamkrieg und breiten Antikriegsprotesten geprägt. Das Smithsonian beschreibt 1969 entsprechend als Jahr, in dem die Mondlandung auf eine von sozialem Streit und Protest gezeichnete Gesellschaft traf.[3]

Auch Wissenschaftlerinnen und Wissenschaftler stellten die politische Neutralität ihrer Arbeit infrage. Am 4. März 1969 unterbrachen Hunderte Lehrende und Studierende am MIT ihre Forschung, um über die Beteiligung von Wissenschaft am Vietnamkrieg zu diskutieren.[4]

In derselben Epoche konnte Technik deshalb sehr verschiedene Bedeutungen tragen: Sie bewies mit Apollo eine kaum vorstellbare Leistungsfähigkeit. Sie war Teil militärischer und geopolitischer Macht. Sie nährte populäre Zukunftsbilder und wurde zugleich Gegenstand politischen Misstrauens.

Vor diesem Hintergrund wirkt die Perzeptron-Kritik beinahe gegenläufig. Während die Raumfahrt öffentlich eine Grenze sprengte, formulierten Minsky und Papert Grenzen einer viel versprochenen Maschinenidee. Das ist keine kausale Verbindung. Es ist ein Zeitfenster. Es erinnert daran, dass wissenschaftliche Ernüchterung und technischer Triumph gleichzeitig stattfinden können.

Der KI-Winter hat mehr als eine Ursache

Der Ausdruck „KI-Winter“ fasst Phasen sinkender Erwartungen und Finanzierung zusammen. Er wird häufig so verwendet, als habe die gesamte künstliche Intelligenz gleichzeitig denselben Frost erlebt.

Tatsächlich unterschieden sich Regionen, Institutionen und Teilgebiete. Enttäuschte Versprechen, begrenzte Rechner, schwierige Wissensakquisition, politische Evaluierungen und das Ausbleiben erwarteter Produkte spielten in verschiedenen Perioden unterschiedliche Rollen.

Für neuronale Netze war *Perceptrons* ein wichtiger Teil der Geschichte. Es ist jedoch methodisch unzulässig, daraus eine monokausale Kette zu bilden:

> *Buch erscheint → Finanzierung endet → neuronale Netze verschwinden.*

Zwischen diesen Pfeilen liegen Entscheidungen, Interessen und konkurrierende Evidenz. Das Buch selbst belegt seinen mathematischen Inhalt, nicht die gesamte institutionelle Wirkung. Für diese Wirkung werden Wissenschaftsgeschichte, Archivmaterial und soziologische Rekonstruktion benötigt.

Die Unterscheidung ist mehr als akademische Sorgfalt. Wer eine komplexe Abkühlung auf einen einzigen Fehler reduziert, lernt die falsche Lektion für die Gegenwart.

Die Grenze wird zum Richtungsweiser

Eine sichtbare Grenze kann Forschung beenden. Sie kann sie auch umlenken.

Wenn eine einzelne lineare Schicht nicht genügt, gibt es mehrere Reaktionen:

- bessere Eingabemerkmale von Hand entwerfen,
- andere Modellfamilien verwenden,
- symbolische Repräsentationen und Regeln entwickeln,
- probabilistische Verfahren verfolgen,
- mehrschichtige Netze bauen,
- oder ein Lernverfahren für interne Repräsentationen suchen.

Welche Reaktion gewinnt, hängt nicht allein von theoretischer Eleganz ab. Sie hängt davon ab, welche Alternative auf vorhandener Hardware, mit verfügbaren Daten und in bestehenden Institutionen produktiv gemacht werden kann.

Die Perzeptron-Grenze war deshalb kein geschlossenes Tor. Sie war eine Kreuzung. Eine wichtige Verkehrsrichtung führte zunächst weg von neuronalen Netzen.

Die soziale Lebensdauer einer Kritik

Technische Argumente besitzen eine soziale Lebensdauer. Ein bewiesenes Ergebnis bleibt gültig, aber seine Bedeutung kann sich ändern, wenn die umgebende Technik wechselt.

Die lineare Grenze eines einfachen Perzeptrons verschwand nicht durch Backpropagation. Sie wurde umgangen, indem andere Architekturen praktisch trainierbar wurden. Die Kritik blieb richtig; ihr Status als Urteil über die Zukunft verlor an Kraft.

Dieses Muster wiederholt sich:

- Eine Methode scheitert unter den Kosten einer Epoche.
- Das Scheitern wird als Eigenschaft der Idee gespeichert.
- Spätere Hardware oder Algorithmen verändern die Kosten.
- Die alte Idee erscheint plötzlich in neuem Licht.

Sara Hookers Hardware-Lottery beschreibt genau diese Gefahr: Eine Forschungsrichtung kann als unproduktiv erscheinen, weil sie schlecht zu den verfügbaren Systemen passt, und dieser lokale Verlust wird mit grundsätzlicher Unterlegenheit verwechselt.[5]

Nicht jede vergessene Idee ist ein verborgenes Juwel. Doch jede historische Niederlage muss daraufhin geprüft werden, welche Randbedingungen im Urteil stecken.

Minsky nicht zum Gegenspieler machen

Marvin Minsky eignet sich schlecht als eindimensionaler Gegner lernender Systeme. Seine Arbeit reichte weit über das Perzeptron hinaus; später beschrieb er Intelligenz selbst als Zusammenspiel vieler begrenzter Prozesse. Seymour Papert prägte zugleich Konstruktionismus und das Lernen mit Computern.

Die Personalisierung „Minsky gegen neuronale Netze“ verengt daher nicht nur die Geschichte des Connectionismus. Sie verengt auch Minsky und Papert.

Für dieses Buch ist ihre Rolle präziser und interessanter:

Sie zeigen, wie eine notwendige Kritik an einer überdehnten technischen Verheißung zu einer Bruchfolge werden kann, deren institutionelle Bedeutung größer ist als der formale Aussageumfang.

Genau diese Möglichkeit muss Project Ant Colony für sich selbst ernst nehmen. Auch dieses Projekt verbindet eine auffällige Metapher, alte Hardware und ein trainiertes Entscheidungsmodell. Es wäre leicht, aus einer begrenzten Forschungsfrage die Verheißung einer allgemeinen alternativen KI zu machen.

Die Perzeptron-Kontroverse warnt:

> *Je größer die Erzählung, desto genauer muss die Reichweite des Experiments begrenzt werden.*

Eine offene Rechnung

Am Ende des Kapitels bleibt eine technische Rechnung offen.

Mehrschichtige Netze besitzen einen größeren Darstellungsraum. Doch wie soll ein Fehler an der Ausgabe in sinnvolle Änderungen weit entfernter interner Gewichte übersetzt werden?

Diese Frage ist das *credit assignment problem*: Welcher innere Beitrag trägt wie viel Verantwortung für das beobachtete Ergebnis?

Die Kettenregel der Analysis war bekannt. Verfahren der Sensitivitätsberechnung existierten in anderen Feldern. Der entscheidende nächste Bruch bestand nicht darin, Mathematik aus dem Nichts zu erfinden. Er bestand darin, eine berechenbare, anschlussfähige und überzeugende Form des Lernens interner Repräsentationen zu etablieren.

Damit beginnt die Geschichte der Rückpropagierung.

Quellenhinweise

1 Marvin Minsky und Seymour A. Papert, *Perceptrons: An Introduction to Computational Geometry*, Cambridge, Massachusetts: MIT Press, 1969.

2 Mikel Olazaran, „A Sociological Study of the Official History of the Perceptrons Controversy“, *Social Studies of Science* 26, Nr. 3 (1996), 611-659, DOI: 10.1177/030631296026003005.

3 Smithsonian Institution, „1969: A Year in the Collections“, zu Apollo 11, Woodstock, sozialen Konflikten und Antikriegsprotesten.

4 National Museum of American History, „Science and Political Protest“, zum Forschungsstreik am MIT am 4. März 1969.

5 Sara Hooker, „The Hardware Lottery“, *Communications of the ACM* 64, Nr. 12 (2021), 58-65, DOI: 10.1145/3467017.

KAPITEL 7

Tiefe wird wieder trainierbar

Der Fehler reist rückwärts

Rückpropagierung öffnet den Trainingsweg zu internen Repräsentationen, lange bevor daraus ein industrieller Standard wird.

Ein mehrschichtiges Netz erzeugt seine Ausgabe in Vorwärtsrichtung. Eingaben werden in einer Schicht transformiert, die resultierenden Aktivierungen in der nächsten, bis eine Vorhersage entsteht.

Das Lernen muss die Gegenrichtung nehmen.

Aus der Abweichung zwischen Vorhersage und Ziel wird berechnet, wie stark jedes Gewicht zum Fehler beigetragen hat. Die Kettenregel zerlegt die Wirkung einer inneren Änderung entlang der Rechenfolge. Der Fehler „reist“ rückwärts durch das Netz; die Gewichte werden so angepasst, dass sich die Zielgröße voraussichtlich verbessert.

Diese Beschreibung klingt heute elementar. 1986 zeigten David E. Rumelhart, Geoffrey E. Hinton und Ronald J. Williams in einer kurzen Nature-Arbeit, wie Rückpropagierung nützliche interne Repräsentationen in Netzen mit verborgenen Einheiten lernen konnte.[1]

Der historische Bruch war nicht die Erfindung der Kettenregel. Er war die überzeugende Verbindung von mehrschichtiger Architektur, differenzierbarer Berechnung, Lernregel und experimenteller Demonstration.

Keine Schöpfung aus dem Nichts

Backpropagation besitzt keine einfache Erfinderbiografie.

Gradienten- und Adjungiertenverfahren, automatische Differentiation und Sensitivitätsanalyse haben frühere Entwicklungslinien in Mathematik, Regelungstechnik und Informatik. Paul Werbos beschrieb in seiner Harvard-Dissertation von 1974 die Anwendung rückwärts gerichteter Differentiation auf lernende Systeme.[2] Weitere Arbeiten näherten sich verwandten Verfahren aus anderen Richtungen.

Rumelhart, Hinton und Williams bleiben dennoch zentral. Historische Bedeutung entsteht nicht nur durch die früheste bekannte Formulierung. Sie entsteht auch durch Klarheit, Zeitpunkt, experimentelle Wirkung und Anschlussfähigkeit an eine entstehende Gemeinschaft.

Das Paper von 1986 machte sichtbar, dass verborgene Einheiten nicht nur zusätzliche Kapazität bereitstellen. Sie konnten während des Trainings Merkmale entwickeln, die für die Aufgabe nützlich waren.

Damit veränderte sich die Rolle des Menschen. Merkmalsentwurf blieb wichtig, aber ein Teil der Repräsentation konnte aus Daten gelernt werden.

Die gelernte Mitte

Beim einfachen Perzeptron ist der Weg zwischen Eingabe und Entscheidung kurz. Die Gewichte bilden unmittelbar eine Grenze.

Ein mehrschichtiges Netz besitzt eine Mitte, deren Bedeutung nicht von außen vorgegeben werden muss. Interne Einheiten können Kombinationen von Eingabemerkmalen darstellen. Weitere Schichten kombinieren diese Repräsentationen erneut.

Diese gelernte Mitte ist eine der folgenreichsten Ideen moderner KI.

Sie erlaubt, Aufgaben nicht ausschließlich durch menschlich definierte Zwischenbegriffe zu lösen. Das System entwickelt Repräsentationen, die durch die Zielfunktion nützlich werden. Daraus folgt weder, dass diese Repräsentationen wahr, verständlich oder biologisch plausibel sind. Aber sie erweitern den praktischen Suchraum erheblich.

Der Preis ist eine neue Form der Undurchsichtigkeit. Je mehr innere Transformationen gelernt werden, desto schwerer ist es, das Ergebnis als Folge weniger expliziter Regeln zu erklären.

Trainierbar bedeutet nicht leicht

Die Überschrift dieses Kapitels ist absichtlich vorsichtig: Tiefe wird *wieder* trainierbar, nicht automatisch beherrscht.

Backpropagation liefert Gradienten. Sie garantiert nicht, dass ein tiefes Netz schnell, stabil oder mit begrenzten Daten zu einer guten Lösung konvergiert.

Frühe tiefe Netze trafen auf mehrere Grenzen:

- Gradienten konnten über viele Schichten sehr klein oder sehr groß werden.
- Rechenleistung und Speicher begrenzten Netzgröße und Experimentzahl.
- Initialisierung und Aktivierungsfunktionen erschwerten stabiles Training.
- Große, gelabelte Datensätze fehlten für viele Aufgaben.
- Werkzeuge für automatische Differentiation und effiziente lineare Algebra waren nicht als einheitlicher ML-Stack verfügbar.
- Ein gutes Ergebnis auf einer kleinen Aufgabe bewies noch keine Skalierbarkeit.

Die algorithmische Öffnung von 1986 benötigte fast drei Jahrzehnte technischer und institutioneller Bruchfolgen, bevor Deep Learning zum dominanten Forschungsprogramm wurde.

Die Zwischenzeit

Zwischen der connectionistischen Wiederbelebung der 1980er Jahre und AlexNet lag keine leere Wartehalle.

Convolutional Networks wurden für Handschrifterkennung entwickelt. Rekurrente Netze, Boltzmann-Maschinen, Support-Vector-Machines, probabilistische Modelle und Ensembles konkurrierten. Forschende arbeiteten an Initialisierung, Regularisierung, Aktivierungsfunktionen, Vortraining und Optimierung. Hardware wurde schneller. Digitale Datensätze wuchsen.

Die spätere Erzählung „Backpropagation plus GPU gleich Deep Learning“ faltet diese Zwischenzeit zusammen. Sie unterschätzt, wie viele lokale Lösungen notwendig waren, bevor ein großer sichtbarer Durchbruch entstehen konnte.

Eine bekannte Idee gewinnt nicht einfach, sobald genug FLOPS verfügbar sind. Sie muss in Software übersetzt, numerisch stabilisiert, an Aufgaben gebunden und in einem Messsystem überzeugend werden.

Software als Verstärker

Backpropagation besitzt eine Eigenschaft, die später perfekt zu modernen Frameworks passen sollte: Sie kann aus einem Rechengraphen systematisch abgeleitet werden.

Wenn jede Operation ihre lokale Ableitung kennt, lässt sich der Gesamtgradient durch den Graphen zusammensetzen. Automatische Differentiation verwandelt damit eine mühsame und fehleranfällige Herleitung in Infrastruktur.

Diese Automatisierung vergrößert nicht nur die Geschwindigkeit. Sie verändert, wer experimentieren kann. Forschende können neue Architekturen aus bekannten Operationen zusammensetzen, ohne jeden Gradienten von Hand zu implementieren.

Die Bruchfolge besteht also aus mindestens drei Stufen:

1. Rückpropagierung macht das Lernen interner Gewichte algorithmisch zugänglich.
2. Automatische Differentiation macht Rückpropagierung für neue Rechengraphen programmatisch zugänglich.
3. Beschleunigte Tensorbibliotheken machen große Rechengraphen praktisch ausführbar.

Erst die Verbindung erzeugt den späteren Standard.

Die Hardware wählt die Form der Tiefe

Nicht jede tiefe Architektur ist gleich gut parallelisierbar.

Modelle mit großen dichten Matrixmultiplikationen oder Faltungen lassen sich anders auf Beschleuniger abbilden als Modelle mit stark verzweigter Logik, unregelmäßigen Speicherzugriffen oder vielen kleinen abhängigen Operationen.

Sobald GPU-Training zum leistungsfähigen Standard wird, erhalten Architekturen einen Vorteil, deren Operationen diesen Körper ausnutzen. Der Gradient bleibt mathematisch korrekt; die Geschwindigkeit seiner Berechnung wird zum Auswahlmechanismus.

Hier beginnt der Beschleuniger-Bias nicht erst bei CUDA. Er ist bereits in der Frage angelegt, welche Formen von Tiefe oft genug trainiert werden können, um verbessert zu werden.

Die Lektion für den Decision Compiler

Backpropagation löst ein Zuordnungsproblem: Wie wird ein globaler Fehler auf viele lokale Parameter verteilt?

Project Ant Colony steht vor einem strukturell verwandten, aber nicht identischen Problem: Wie wird ein globales Ziel - Zeit bis Qualität unter Sicherheitsgrenzen - auf lokale Entscheidungen über Knoten, Kommunikation, Batchgrößen, Synchronisation und Zeitfenster bezogen?

Es wäre verführerisch, dafür ein einziges großes Modell end-to-end lernen zu lassen. Die Geschichte mahnt zur Trennung.

Backpropagation funktioniert innerhalb eines klar definierten, differenzierbaren Rechengraphen. Ein reales Cluster enthält Ausfälle, diskrete Regeln, ablaufende Zustände und irreversible Aktionen. Seine Sicherheitsverträge dürfen nicht als weiche Verlustterme behandelt werden.

Das trainierte Modell darf daher den Entscheidungsraum durchsuchen. Die Ausführbarkeit bleibt Aufgabe eines deterministischen Kerns.

Die gelernte Mitte erzeugt Vorschläge. Die harte Grenze entscheidet, was Wirklichkeit werden darf.

Vor dem sichtbaren Durchbruch

Ende der 1980er Jahre war eine entscheidende Grenze verschoben. Interne Repräsentationen mehrschichtiger Netze konnten systematisch gelernt werden.

Doch die neue Methode wartete noch auf eine breitere Passung:

- große und vergleichbare Daten,
- genügend schnelle parallele Hardware,
- effiziente Implementierungen,
- Regularisierung für große Modelle,
- und einen Wettbewerb, in dem der Abstand öffentlich sichtbar wurde.

Diese Passung erhielt 2012 einen Namen, obwohl sie weit mehr als ein einzelnes Modell umfasste: AlexNet.

Quellenhinweise

1 David E. Rumelhart, Geoffrey E. Hinton und Ronald J. Williams, „Learning Representations by Back-Propagating Errors“, *Nature* 323 (1986), 533-536, DOI: 10.1038/323533a0.

2 Paul J. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, Dissertation, Harvard University, 1974.

KAPITEL 8

Daten, Hardware und der AlexNet-Moment

Ein Durchbruch mit vielen Eltern

ImageNet, Architektur, Regularisierung und GPU-Code greifen zu einem sichtbaren Bündelbruch ineinander.

Im Herbst 2012 veränderte ein Wettbewerb die Erzählung künstlicher Intelligenz.

Alex Krizhevsky, Ilya Sutskever und Geoffrey Hinton reichten ein tiefes Convolutional Network beim ImageNet-Wettbewerb ein. Das Modell erreichte im ILSVRC-2012-Test eine Top-5-Fehlerrate von 15,3 Prozent; der zweitbeste Beitrag lag bei 26,2 Prozent.[1] Der Abstand war groß genug, um nicht als kleiner methodischer Fortschritt zu erscheinen.

AlexNet wurde zum Symbol des Deep-Learning-Durchbruchs.

Symbole verdichten. Sie machen Geschichte sichtbar, aber sie ziehen viele Ursachen in ein Bild zusammen. Bei AlexNet ist die Versuchung besonders groß, den Moment auf einen Faktor zu reduzieren: das tiefe Netz, die GPU oder den großen Datensatz.

Der tatsächliche Bruch war die Passung aller drei - ergänzt um Regularisierung, Implementierung und Wettbewerb.

ImageNet machte Größe messbar

Ein leistungsfähiges Lernverfahren benötigt eine Aufgabe, an der seine Leistung sichtbar wird.

ImageNet wurde als groß angelegte, hierarchisch organisierte Bilddatenbank entwickelt. Die Kategorien orientierten sich an WordNet; zu vielen Begriffen wurden zahlreiche Bilder gesammelt.[2] Der Datensatz verschob damit eine Grenze der visuellen Objekterkennung. Statt auf kleinen, relativ homogenen Sammlungen zu arbeiten, konnten Modelle an einer sehr großen Zahl von Kategorien und variierenden Bildern geprüft werden.

Die ImageNet Large Scale Visual Recognition Challenge formte daraus eine Messordnung. Ein gemeinsamer Datensatz, definierte Aufgaben und vergleichbare Metriken machten Unterschiede öffentlich und wiederholbar.

Diese institutionelle Leistung ist leicht zu unterschätzen. Ohne ein glaubwürdiges gemeinsames Messsystem kann ein besseres Modell als lokaler Einzelfall erscheinen. Mit einem Wettbewerb wird der Abstand zur Rangliste.

ImageNet lieferte also nicht bloß mehr Trainingsmaterial. Es schuf eine Bühne, auf der Skalierung eine erkennbare Belohnung erhielt.

Die Architektur war kein neutraler Behälter

AlexNet besaß rund 60 Millionen Parameter und acht Schichten mit lernbaren Gewichten: fünf Faltungs- und drei vollständig verbundene Schichten. Das Modell nutzte unter anderem Rectified Linear Units, lokale Antwortnormalisierung, überlappendes Pooling, Datenaugmentation und Dropout.[1]

Jeder dieser Bestandteile beantwortete eine konkrete Grenze.

Faltungen nutzten räumliche Struktur und Gewichtsteilung. ReLUs beschleunigten in den berichteten Experimenten das Training gegenüber sättigenden Aktivierungen. Datenaugmentation und Dropout wirkten Überanpassung entgegen. Die Aufteilung auf zwei GPUs umging Speichergrenzen und ermöglichte ein Modell, das auf einer einzelnen verwendeten Karte nicht in gleicher Form Platz gehabt hätte.

Das Ergebnis war deshalb nicht „ein altes Netz, nur größer“. Architektur und Training waren auf die Möglichkeit größerer Berechnung abgestimmt.

Skalierung ist nie bloß eine Zahl. Sie verändert, welche technischen Probleme gelöst werden müssen.

Zwei Grafikkarten und fünf bis sechs Tage

Die Autoren trainierten das Netz auf zwei NVIDIA GTX 580 mit jeweils drei Gigabyte Speicher. Das Training dauerte fünf bis sechs Tage. Sie schrieben ausdrücklich, dass die Größe des Netzes vor allem durch den verfügbaren GPU-Speicher und die tolerierte Trainingszeit begrenzt war.[1]

Diese Angaben sind mehr als historische Kuriositäten. Sie zeigen den Körper des Durchbruchs.

Die GPUs machten eine große Zahl ähnlicher numerischer Operationen schnell ausführbar. Der CUDA-basierte Code erlaubte eine hochoptimierte Implementierung. Gleichzeitig prägten Speichergröße und Geräteaufteilung die Architektur. Ein Teil der Verbindungen wurde so organisiert, dass Zwischenkommunikation zwischen den beiden Karten begrenzt blieb.

Hardware war damit weder bloßer Motor noch neutraler Untergrund. Sie war eine Entwurfsbedingung.

Warum nicht schon früher?

Convolutional Networks, Gradientenverfahren und parallele Prozessoren existierten vor 2012. Warum entstand der Bruch gerade dann?

Weil mehrere Bruchfolgen zusammenliefen:

- Digitale Bildbestände und verteilte Annotation ermöglichten einen großen Datensatz.
- Ein Wettbewerb machte Fortschritt in einer klaren Metrik sichtbar.
- GPUs boten hohe parallele Rechenleistung und Speicherbandbreite.
- CUDA und spezialisierter Code machten diese Leistung für allgemeine Berechnungen zugänglich.
- Verbesserungen bei Aktivierungen, Regularisierung und Training stabilisierten große Netze.
- Eine Forschungsgruppe verband diese Komponenten in einem reproduzierbaren System.

Jeder Punkt allein wäre unzureichend gewesen. Ein großer Datensatz ohne trainierbares Modell liefert keine gute Erkennung. Ein großes Modell ohne Regularisierung überpasst. Eine GPU ohne geeigneten Kernel wartet. Ein leistungsfähiges System ohne gemeinsamen Benchmark bleibt eine Demonstration unter vielen.

Der AlexNet-Moment war ein Bündelbruch.

Der Abstand erzeugt Gewissheit

Kleine Verbesserungen lassen Raum für Zweifel. Vielleicht ist die Implementierung besser abgestimmt, vielleicht die Metrik günstig gewählt, vielleicht der Effekt nicht robust.

Ein großer Abstand verändert die soziale Verarbeitung eines Ergebnisses. Er rechtfertigt Investitionen, zieht Forschende an und verschiebt die Beweislast. Nach AlexNet musste nicht mehr primär erklärt werden, warum man tiefe Convolutional Networks untersuchte. Erklären musste sich zunehmend, wer auf dem wichtigen Bildbenchmark ohne sie konkurrenzfähig bleiben wollte.

Diese Umkehrung ist ein Kennzeichen des Paradigmenwechsels.

Die Methode wird vom riskanten Spezialweg zum Default. Werkzeuge, Lehrveranstaltungen und Förderentscheidungen richten sich neu aus. Andere Aufgaben werden so formuliert, dass derselbe Ansatz geprüft werden kann.

Der Durchbruch verändert damit nicht nur die Antwort auf ImageNet. Er verändert die nächsten Fragen.

Die Bruchfolge des Erfolgs

Nach 2012 wuchs ein beschleunigender Kreislauf.

Mehr Forschende entwickelten tiefe Modelle. Frameworks vereinfachten automatische Differentiation und GPU-Ausführung. Hersteller optimierten Bibliotheken und Hardware für neuronale Operationen. Größere Datensätze und Benchmarks belohnten zusätzliche Kapazität. Unternehmen sahen Anwendungen in Vision, Sprache und Empfehlungssystemen. Cloudplattformen machten Beschleuniger mietbar.

Der Erfolg reduzierte die Kosten des nächsten Erfolgs.

Diese Bruchfolge erklärt einen Teil der Geschwindigkeit, mit der Deep Learning andere Felder erfasste. Die Methode musste nicht in jeder Domäne ihre gesamte Infrastruktur neu erfinden. Ein wachsender Stack wurde übertragbar.

Genau hier wird CUDA historisch wichtiger als die zwei Karten eines einzelnen Experiments. Es verband viele Gerätegenerationen, Bibliotheken und Frameworks zu einer fortlaufenden Entwicklungsplattform.

Das Gegenbild: DistBelief

Im selben Jahr beschrieben Forschende bei Google DistBelief, ein System zum Training großer neuronaler Netze auf verteilten CPU-Clustern.[3] Modell- und Datenparallelität, replizierte Modelle und ein partitionierter Parameter-Server ermöglichten Experimente auf Tausenden Maschinen.

DistBelief war nicht das Gegenargument zu AlexNet. Die Arbeiten lösten andere Aufgaben in anderen Infrastrukturen. Gemeinsam zeigen sie jedoch, dass 2012 mehr als ein Skalierungspfad real war.

Der GPU-Pfad bündelte enorme lokale Parallelität in vergleichsweise wenigen Geräten. Der Cluster-Pfad verteilte Arbeit über sehr viele allgemeine Maschinen und musste Kommunikation, Asynchronität und Ausfälle expliziter behandeln.

Die tatsächliche Geschichte behielt beide Linien. Doch ihre Sichtbarkeit und Zugänglichkeit entwickelten sich unterschiedlich.

Was AlexNet nicht beweist

AlexNet beweist nicht, dass jede wichtige KI-Aufgabe am besten durch immer größere dichte Netze gelöst wird.

Es beweist nicht, dass GPU-Hardware die einzige mögliche Trägerin tiefer Netze ist.

Es beweist nicht, dass Datenmenge und Rechenleistung Architektur, Regularisierung und sorgfältige Implementierung ersetzen.

Es zeigt etwas engeres und historisch dennoch Gewaltiges:

> *Wenn eine Modellklasse, ein großer Datensatz, geeignete Lernverfahren und eine zugängliche parallele Plattform ineinandergreifen, kann aus einer lange bekannten Forschungsrichtung in kurzer Zeit eine dominante Arbeitsordnung werden.*

Die nächste Frage lautet daher nicht, ob GPUs „schuld“ am Deep Learning sind. Sie lautet, wie aus programmierbaren Grafikprozessoren ein Ökosystem wurde, das diese Passung über viele Aufgaben hinweg industrialisierte.

Quellenhinweise

1 Alex Krizhevsky, Ilya Sutskever und Geoffrey E. Hinton, „ImageNet Classification with Deep Convolutional Neural Networks“, *Advances in Neural Information Processing Systems 25* (2012), 1097-1105.

2 Jia Deng et al., „ImageNet: A Large-Scale Hierarchical Image Database“, *IEEE Conference on Computer Vision and Pattern Recognition* (2009), 248-255, DOI: 10.1109/CVPR.2009.5206848.

3 Jeffrey Dean et al., „Large Scale Distributed Deep Networks“, *Advances in Neural Information Processing Systems 25* (2012), 1223-1231.

KAPITEL 9

CUDA und die Industrialisierung eines Paradigmas

Der Unterschied zwischen Leistung und Zugang

Aus programmierbaren Grafikprozessoren entsteht ein wiederholbarer Entwicklungs- und Bibliotheksstack.

Eine Grafikkarte kann sehr viele Rechenoperationen ausführen. Daraus folgt nicht, dass eine Forscherin sie für ein beliebiges Problem produktiv nutzen kann.

Vor CUDA wurden Grafikprozessoren zunehmend programmierbar. Allgemeine Berechnungen mussten jedoch oft in die Sprache der Grafikpipeline übersetzt werden: Texturen, Fragmente, Shader und Renderdurchläufe. Das Silizium war vorhanden; der Zugang blieb umständlich.

Brook for GPUs zeigte 2004 einen wichtigen Zwischenweg. Eine Stream-Programmiersprache und ihr Compiler abstrahierten Teile der Grafikdetails und behandelten die GPU als datenparallelen Coprozessor.[1] Die Arbeit belegt, dass General-Purpose Computing on GPUs nicht mit CUDA begann.

Sie zeigt zugleich die eigentliche Engstelle: Rechenleistung wird erst dann historisch wirksam, wenn sie in eine beherrschbare Programmierform übersetzt wird.

Der Highway entsteht

NVIDIA führte CUDA 2006 als allgemeine parallele Plattform und Programmiermodell ein. Die offizielle Dokumentation beschreibt den Schritt als Möglichkeit, beliebige Rechenlasten unabhängig von Grafik-APIs auf den Durchsatz der GPU zugreifen zu lassen.[2]

Der praktische Unterschied lag in einer neuen Arbeitsordnung:

- Entwickler konnten Kernel in einer C-nahen Sprache schreiben.
- Ein Compiler übersetzte Host- und Gerätecode.
- Threads, Blöcke und Grids bildeten ein verständliches Parallelitätsmodell.

- Speicherbereiche und Synchronisationsregeln wurden explizit.
- Eine fortlaufende Plattform verband neue Hardwaregenerationen mit bestehendem Code.

CUDA machte GPU-Programmierung nicht automatisch leicht. Gute Kernel verlangten Wissen über Speicherzugriffe, Parallelität, Auslastung und Gerätearchitektur. Aber die Schwierigkeit wechselte ihre Art. Entwickler mussten ihr Problem nicht länger als Grafikproblem verkleiden.

Aus einer Nebenstraße wurde ein Highway.

Eine API reicht nicht

Wäre CUDA nur eine Programmierschnittstelle geblieben, hätte es die KI-Geschichte kaum in gleicher Weise geprägt.

Die Industrialisierung entstand durch Schichten.

Ganz unten stand die Hardware: immer leistungsfähigere GPUs mit hoher Speicherbandbreite und wachsender Eignung für allgemeine numerische Lasten.

Darüber lagen Treiber, Compiler und ein stabiles Programmiermodell. Bibliotheken wie cuBLAS, cuFFT und später cuDNN boten optimierte Standardoperationen. cuDNN stellte hoch abgestimmte Primitive für Vorwärts- und Rückwärtsoperationen neuronaler Netze bereit, darunter Faltungen, Aktivierungen und Pooling.[3]

Frameworks konnten diese Bibliotheken nutzen, ohne dass jede Forschungsgruppe eigene Kernel für jede Hardwaregeneration schreiben musste. Automatische Differentiation verband Modellbeschreibung und Gradienten. Paketverwaltung, Dokumentation, Beispiele und Community-Wissen senkten weitere Hürden.

Schließlich machten Cloudangebote dieselbe Logik als mietbare Infrastruktur verfügbar.

Das Produkt war nicht mehr „eine schnelle Grafikkarte“. Es war eine Entwicklungskette von der Modellidee bis zur ausführbaren Operation.

Industriell bedeutet wiederholbar

Industrialisierung meint hier keine Massenfertigung allein. Sie meint die Standardisierung eines Forschungsprozesses.

Eine neue Architektur konnte aus bekannten Tensoroperationen zusammengesetzt werden. Der Gradient entstand automatisch. Bibliotheken wählten schnelle Kernel. Treiber adressierten die Geräte. Verteilte Laufzeiten koordinierten mehrere Beschleuniger. Messwerkzeuge zeigten Auslastung und Speicher.

Diese Schichtung verringerte den Anteil einmaliger Handarbeit.

Das war wissenschaftlich außerordentlich produktiv. Mehr Menschen konnten Ideen testen. Ergebnisse wurden leichter reproduzierbar. Verbesserungen einer Bibliothek beschleunigten viele Modelle zugleich. Neue Hardware konnte einen bestehenden Stack schneller machen, ohne dass jede Anwendung neu geschrieben werden musste.

> *CUDA industrialisierte nicht die Idee des neuronalen Netzes. Es industrialisierte den Weg von einer bestimmten Klasse von Ideen zu schnellen Experimenten.*

Der Stack wählt mit

Jeder Stack besitzt eine bevorzugte Form.

GPUs sind besonders stark, wenn viele ähnliche Operationen parallel ausgeführt werden, genügend Arbeit vorhanden ist und Datenbewegung gut organisiert werden kann. Große Faltungen und Matrixmultiplikationen passen zu diesem Profil. Stark verzweigte Programme, kleine serielle Schritte oder unregelmäßige Speicherzugriffe können die Hardware schlechter auslasten.

Sobald Framework und Bibliotheken den passenden Weg extrem bequem machen, verändert sich der Preis wissenschaftlicher Ideen.

Ein Modell, das sich aus gut unterstützten Operationen zusammensetzt, erbt Jahre von Optimierung. Eine ungewöhnliche Operation beginnt mit einem eigenen Kernel, eigenen Fehlern und ungewisser Portabilität. Selbst wenn sie algorithmisch interessant ist, kostet ihr erster überzeugender Versuch mehr.

Das ist der Beschleuniger-Bias in seiner industriellen Form.

Er entsteht nicht durch eine geheime Vorgabe. Er entsteht durch rationale Wiederverwendung. Forschende verwenden Werkzeuge, die funktionieren. Hersteller optimieren Operationen, die nachgefragt werden. Frameworks stabilisieren Pfade, auf denen viele Nutzer arbeiten.

Die Richtung entsteht aus der Summe vernünftiger Entscheidungen.

Der offene Gegenpfad

2008 veröffentlichte die Khronos Group OpenCL 1.0 als offenen, lizenzfreien Standard für parallele Programmierung über CPUs, GPUs und andere Prozessoren.[4]

Die Gegenüberstellung ist lehrreich. CUDA bot eine eng integrierte Plattform aus einer Hand. OpenCL versprach Hersteller- und Gerätevielfalt unter einer gemeinsamen Spezifikation.

Offenheit löst jedoch nicht automatisch das Leistungsproblem. Unterschiedliche Geräte besitzen verschiedene Speicherhierarchien, Treiberqualitäten und Optimierungsanforderungen. Ein portabler Kernel kann korrekt auf vielen Zielsystemen laufen und dennoch auf keinem optimal sein.

CUDA gewann seinen praktischen Vorteil nicht nur durch Exklusivität, sondern durch Integration. Hardware, Compiler, Bibliothek und Dokumentation konnten gemeinsam entwickelt werden.

Der Preis dieser Integration ist Pfadbindung. Je mehr Code, Wissen und Modelle auf einem Stack aufbauen, desto teurer wird der Wechsel.

Bindung ist nicht bloß Lock-in

Der Begriff *Lock-in* klingt, als seien Nutzer in eine Falle geraten. Das beschreibt nur einen Teil.

Bindung entsteht auch aus produktivem Vertrauen. Ein Labor weiß, welche Karten funktionieren. Mitarbeitende kennen die Werkzeuge. Fehler lassen sich diagnostizieren. Publizierter Code setzt dieselbe Plattform voraus. Ein Cloudanbieter bietet passende Instanzen. Die nächste Beschaffung ist unter diesen Bedingungen keine freie Wahl zwischen abstrakten Alternativen.

Diese Pfadabhängigkeit kann vernünftig und dennoch richtungsbildend sein.

Sie erklärt, warum ein konkurrierender Stack nicht nur technisch „fast so gut“ sein darf. Er muss die angesammelte Sicherheit, Geschwindigkeit und soziale Infrastruktur des dominanten Pfades überwinden.

Vom Werkzeug zum Paradigma

Ein Werkzeug wird zum Teil eines Paradigmas, wenn seine Annahmen in die Forschungsfrage eingehen.

Nach der Etablierung des GPU-Stacks wurde häufig nicht mehr gefragt, welche Hardwareform zu einer Aufgabe passt. Gefragt wurde, wie ein Modell auf einer oder mehreren GPUs skaliert. Speichergrenzen führten zu Model Parallelism, Checkpointing und niedrigeren Präzisionen. Kommunikationsgrenzen führten zu optimierten Kollektiven. Architektur und Infrastruktur entwickelten sich gemeinsam weiter.

Das ist keine Kritik an diesen Lösungen. Sie sind technische Meisterleistungen.

Der historische Punkt lautet: Die Maschine war nicht länger eine austauschbare Ausführungsplattform. Sie wurde zur stillen Voraussetzung, um die herum neue Algorithmen entworfen wurden.

Die neue Grenze

Der Highway entfernte Reibung und erzeugte dadurch Verkehr.

Modelle wurden größer. Trainingsläufe nutzten mehr Geräte. Daten- und Modellparallelität verlangten schnelle Verbindungen. Rechenzentren, Energieversorgung und Kapital wurden zu zentralen Bedingungen. Die Zugänglichkeit einzelner GPUs führte nicht automatisch zu gleich verteiltem Zugang an der Spitze.

Die nächste Grenze entstand aus dem Erfolg:

> *Wenn zusätzliche Rechenleistung zuverlässig in bessere Ergebnisse übersetzt werden kann, wird die Fähigkeit zur Skalierung selbst zum Wettbewerbsvorteil.*

Damit beginnt das Zeitalter der Foundation Models und des Beschleuniger-Narrativs.

Quellenhinweise

1. Ian Buck et al., „Brook for GPUs: Stream Computing on Graphics Hardware“, *ACM Transactions on Graphics* 23, Nr. 3 (2004), 777-786, DOI: 10.1145/1015706.1015800.
2. NVIDIA, *CUDA Programming Guide*, Abschnitt „Introduction“; Einführung der Compute Unified Device Architecture im Jahr 2006.
3. NVIDIA, „Accelerate Machine Learning with the cuDNN Deep Neural Network Library“, 2014; cuDNN-Dokumentation zu GPU-beschleunigten DNN-Primitiven.
4. Khronos Group, „OpenCL 1.0 Released“, 8. Dezember 2008.

KAPITEL 10

Foundation Models und das Beschleuniger-Narrativ

Ein Modell wird zur Grundlage

Skalierung und Foundation Models verwandeln den Beschleunigerpfad in eine institutionelle Arbeitsordnung.

Lange wurde maschinelles Lernen aufgabenweise organisiert. Ein System klassifizierte Bilder, ein anderes erkannte Sprache, ein drittes übersetzte. Architektur, Daten und Training waren stark an die einzelne Aufgabe gebunden.

Foundation Models veränderten diese Arbeitsordnung. Ein großes Modell wird auf breiten Datenbeständen vortrainiert und anschließend durch Prompts, Feinabstimmung oder zusätzliche Komponenten an viele Aufgaben angepasst. Der Stanford-Bericht, der den Begriff 2021 prägte, betonte zugleich die neue Hebelwirkung und die Risiken der Homogenisierung: Stärken und Fehler eines Basismodells verbreiten sich auf viele nachgelagerte Anwendungen.[1]

Das Modell wird zur Plattform.

Diese Verschiebung ist ohne die vorangegangene Industrialisierung des Beschleunigerstacks kaum zu verstehen. Vortraining verlangt große Mengen wiederholbarer Tensorberechnung. Verteilte Laufzeiten, schnelle Kommunikation, automatische Differentiation und hochoptimierte Kernel machen aus einer Architektur einen monatelang stabilen Produktionsprozess.

Der Transformer und die Parallelität

2017 stellten Vaswani und Kollegen den Transformer als Architektur für Sequenztransduktion vor. Er verzichtete auf die rekurrente Verarbeitung früherer Modelle und stützte sich auf Attention-Mechanismen.[2] Die Autoren hoben neben der Qualität die bessere Parallelisierbarkeit und kürzere Trainingszeit hervor.

Diese Eigenschaft war historisch folgenreich.

Rekurrente Netze verarbeiten eine Sequenz in vielen aufeinander abhängigen Schritten. Transformer können während des Trainings viele Positionen einer Sequenz parallel

bearbeiten. Attention ist rechen- und speicherintensiv, lässt sich aber in große Matrixoperationen übersetzen.

Architektur und Beschleuniger passten erneut zusammen.

Der Transformer war nicht bloß ein Produkt der GPU. Er wurde auch auf TPUs und anderen Systemen trainiert. Doch seine stark tensorisierte Form konnte vom wachsenden Beschleunigerökosystem besonders profitieren. Bessere Hardware machte größere Modelle möglich; größere Modelle rechtfertigten weitere Optimierung des Stacks.

Skalierung wird zur Methode

Empirische Arbeiten zu Skalierungsgesetzen zeigten, dass sich der Sprachmodellverlust über weite Bereiche vorhersagbar mit Modellgröße, Datenmenge und Trainingsrechenleistung verändert.[3] Diese Beobachtung verwandelte Compute von einer Randbedingung in eine Forschungsvariable.

Wenn mehr sorgfältig eingesetzte Rechenleistung verlässlich bessere Modellqualität verspricht, entsteht eine klare Strategie:

1. Datenbestand vergrößern.
2. Modellkapazität erhöhen.
3. Training über mehr Beschleuniger verteilen.
4. Systemengpässe beseitigen.
5. Fähigkeiten auf immer mehr Aufgaben prüfen.

GPT-3 demonstrierte 2020 mit 175 Milliarden Parametern, dass ein stark skaliertes autoregressives Sprachmodell viele Aufgaben durch wenige Beispiele im Kontext bearbeiten konnte, ohne für jede Aufgabe seine Gewichte zu verändern.[4]

Der sichtbare Fortschritt war real. Ebenso real war die neue institutionelle Schwelle. Ein einzelnes Experiment erforderte nun eine Infrastruktur, die nur wenige Organisationen kontrollierten.

Das Beschleuniger-Narrativ

Aus einer erfolgreichen Methode kann eine Erzählung werden:

> *Intelligenzfortschritt ist vor allem eine Frage größerer Modelle, größerer Datenmengen und größerer Beschleunigercluster.*

Das Narrativ ist nicht erfunden. Es stützt sich auf beobachtete Leistungssprünge. Problematisch wird es, wenn aus einem sehr erfolgreichen Pfad der einzig ernsthafte Evolutionsweg wird.

Die Erzählung besitzt mehrere Verkürzungen.

Erstens wird Compute als homogene Menge behandelt. Tatsächlich unterscheiden sich Speicher, Kommunikation, Präzision, Auslastung und Fehlertoleranz.

Zweitens erscheint Algorithmik als relativ stabiler Behälter, der vor allem größer ausgeführt wird. Dabei entscheiden Optimierer, Datenmischung, Tokenisierung, Sparsity, Routing, Regularisierung und Trainingsordnung wesentlich über den Ertrag eines Rechenbudgets.

Drittens verschwindet das Ökosystem. Ein FLOP ist keine verfügbare Forschungsoperation. Zwischen theoretischer Rechenmenge und einem erfolgreichen Lauf liegen Compiler, Kernel, Netzwerke, Personal, Energieversorgung und Fehlersuche.

Viertens wird der Siegerpfad rückwirkend naturalisiert. Weil große dichte Modelle auf Beschleunigerclustern erfolgreich sind, erscheinen kleinere, modulare oder hardwareadaptive Systeme als bloße Nachoptimierung.

Das Korrektiv aus dem eigenen Paradigma

Die Skalierungsforschung selbst widerlegt die grobe Formel „mehr Parameter genügen“.

Hoffmann und Kollegen untersuchten 2022 die compute-optimale Verteilung zwischen Modellgröße und Trainingsdaten. Ihr Chinchilla-Modell mit 70 Milliarden Parametern übertraf unter gleichem Rechenbudget deutlich größere Modelle, weil es auf mehr Daten trainiert wurde.[5]

Der Befund bleibt innerhalb des Transformer- und Beschleunigerparadigmas. Er zeigt dennoch etwas Grundsätzliches:

> *Ein festes Rechenbudget besitzt keinen festen Erkenntnisertrag. Der Algorithmus und die Allokation bestimmen, was aus dem Budget wird.*

Damit kehrt die Leitthese dieses Buches im Zentrum der Skalierung selbst zurück. Algorithmische Verbesserung ist keine dekorative Effizienzmaßnahme. Sie verändert, welche Möglichkeiten ein vorhandener Körper trägt.

Dichte wird zum Default

Viele Foundation Models aktivieren für jede Eingabe einen großen Teil ihrer Parameter. Diese Dichte passt zu hoch optimierten Matrixoperationen. Sie macht die Ausführung regelmäßig und den Stack beherrschbar.

Es existieren Gegenbewegungen: Mixture-of-Experts aktiviert nur Teile eines Modells; Retrieval verschiebt Wissen in externe Speicher; Quantisierung reduziert Zahlenbreiten; Distillation, Pruning und Sparsity verändern den Ausführungsbedarf. Doch häufig werden diese Verfahren als Optimierung eines zuvor dicht gedachten Modells behandelt.

Das Beschleuniger-Narrativ zeigt sich genau in dieser Reihenfolge:

1. Zuerst wird ein leistungsfähiges Modell unter günstigen Beschleunigerbedingungen entwickelt.
2. Danach wird es für kleinere oder abweichende Körper angepasst.

Eine alternative Evolution könnte die Reihenfolge umdrehen:

1. Zuerst werden Speicher, Kommunikation, Verfügbarkeit und Energie als Entwurfsbedingungen festgelegt.
2. Daraus entsteht eine Modell- und Lernform.

Project Ant Colony untersucht diese Umkehrung nicht auf der Skala eines Foundation-Model-Trainings. Sein Cluster ist absichtlich klein. Aber die methodische Frage ist dieselbe.

Konzentration als Bruchfolge

Foundation Models verbinden Modellleistung mit mehreren Formen von Konzentration:

- Rechenkapazität konzentriert sich in großen Clustern.
- Trainingsdaten und Aufbereitungspipelines werden schwer reproduzierbar.
- Systemwissen verteilt sich auf spezialisierte Teams.
- Zugang erfolgt häufig über APIs statt über vollständig prüfbare Artefakte.
- Nachgelagerte Anwendungen übernehmen Entscheidungen eines Basismodells.

Diese Konzentration ist keine einfache Folge von CUDA. TPUs, anwendungsspezifische Chips, Netzwerktechnik, Cloudökonomie und Kapitalmärkte gehören ebenfalls zur Entwicklung. CUDA ist ein wichtiger Verzweigungspunkt, nicht die einzige Ursache.

Gerade deshalb eignet sich das Gegenfaktum „ohne CUDA“. Es entfernt nicht Beschleunigung oder Kapital. Es prüft, wie stark ein bestimmter integrierter Highway die Form und Verteilung der Skalierung geprägt hat.

Was der dominante Pfad verdeckt

Ein dominantes Paradigma widerlegt seine Alternativen nicht vollständig. Es entzieht ihnen Vergleichsgelegenheiten.

Ein kommunikationsarmes Verfahren kann auf einem schnellen Beschleunigerverbund wenig Vorteil zeigen und auf einem langsamen Commodity-Netzwerk entscheidend sein. Ein modularer Spezialist kann auf einem Benchmark gegen ein großes Generalistenmodell verlieren und in einer lokalen Aufgabe das bessere Verhältnis aus Qualität, Energie und Kontrolle besitzen. Ein hardwareadaptiver Compiler kann gegenüber einer perfekt gepflegten Standardbibliothek überflüssig erscheinen und in einer heterogenen Gerätelandschaft zum Kern werden.

Das sind keine Behauptungen über Überlegenheit. Es sind offene Bedingungssätze.

Die reale Entwicklungslinie dieses Buches endet deshalb nicht mit einer Anklage gegen Beschleuniger. Sie endet mit einer methodischen Unruhe:

> *Haben wir nur gelernt, welche KI unter dem erfolgreichsten Stack gewinnt - oder auch, welche KI unter anderen materiellen Bedingungen möglich wäre?*

Um diese Frage zu beantworten, muss die Geschichte ihren tatsächlichen Verlauf für einen Moment verlassen.

Quellenhinweise

1 Rishi Bommasani et al., *On the Opportunities and Risks of Foundation Models*, Stanford Center for Research on Foundation Models, 2021, arXiv:2108.07258.

2 Ashish Vaswani et al., „Attention Is All You Need“, *Advances in Neural Information Processing Systems 30* (2017), 5998-6008.

3 Jared Kaplan et al., „Scaling Laws for Neural Language Models“, arXiv:2001.08361, 2020.

4 Tom B. Brown et al., „Language Models are Few-Shot Learners“, *Advances in Neural Information Processing Systems 33* (2020), 1877-1901.

5 Jordan Hoffmann et al., „Training Compute-Optimal Large Language Models“, arXiv:2203.15556, 2022.

TEIL III

Die Entwicklungslinie, die nicht stattfand

Ein enges CUDA-Gegenfaktum öffnet konkurrierende Welten und andere Formen von KI.

Kapitel 11 bis 14

KAPITEL 11

Darf man Technikgeschichte anders denken?

Die Grenze zwischen Analyse und Roman

Ein strenges Gegenfaktum trennt historische Anker, plausible Szenarien und heute prüfbare Folgen.

„Was wäre geschehen, wenn es CUDA nie gegeben hätte?“

Die Frage klingt nach Science-Fiction. Sie kann leicht zu einer Wunschgeschichte werden: offene Standards gewinnen, kleine Modelle bleiben wichtig, Rechenmacht verteilt sich gerechter und ein kluges Model OS verbindet vielfältige Geräte.

Nichts davon folgt aus der Abwesenheit einer Plattform.

Ein Gegenfaktum ist kein Bericht über eine verborgene Vergangenheit. Es ist ein analytisches Instrument. Es verändert eine Bedingung und fragt, welche Abhängigkeiten dadurch sichtbar werden.

Sein Wert liegt nicht in der Behauptung, die alternative Welt korrekt vorherzusagen. Sein Wert liegt darin, die scheinbare Notwendigkeit der realen Welt zu prüfen.

Der minimale Eingriff

Ein brauchbares technikhistorisches Gegenfaktum beginnt mit einem möglichst engen Eingriff.

Für dieses Buch lautet er:

> *NVIDIA führt 2006 keine allgemein zugängliche CUDA-Plattform ein, die allgemeine Rechenlasten unabhängig von Grafik-APIs auf NVIDIA-GPUs programmierbar macht und sich zu einem integrierten Hardware-, Compiler-, Bibliotheks- und Frameworkökosystem entwickelt.*

Nicht entfernt werden:

- Grafikprozessoren,

- parallele Berechnung,
- neuronale Netze,
- digitale Datenmengen,
- Rechenzentren,
- Spezialchips,
- wirtschaftlicher Wettbewerb,
- oder der Wunsch nach schnellerer KI.

Diese Begrenzung verhindert einen häufigen Fehler. Wer zu viele Bedingungen zugleich verändert, kann jede gewünschte Welt erzeugen. Dann erklärt das Gegenfaktum nicht die Wirkung von CUDA, sondern nur die Vorlieben des Autors.

Die Welt muss widerständig bleiben

Auch in der alternativen Entwicklung gelten Physik und Ökonomie.

Große Matrixoperationen benötigen Rechenzeit. Gewichte und Aktivierungen müssen durch Speicher bewegt werden. Verteiltes Training erzeugt Kommunikation. Chipentwicklung ist teuer. Softwarefragmentierung kostet Arbeitszeit. Institutionen suchen Wettbewerbsvorteile.

Eine Welt ohne CUDA darf diese Widerstände nicht durch moralische Sympathie auflösen.

Offene Standards können fragmentiert und langsam sein. CPU-Cluster können enorme Energie- und Kommunikationskosten erzeugen. FPGAs können schwer programmierbar bleiben. Proprietäre ASICs können Macht stärker konzentrieren als ein verbreiteter GPU-Stack. Algorithmische Sparsamkeit kann Qualität verlieren oder aufwendiger sein als zusätzliche Hardware.

Ein Gegenfaktum wird erst ernsthaft, wenn es unerwünschte Folgen zulässt.

Reale Nebenpfade statt erfundener Technik

Die alternative Welt dieses Buches wird nicht aus beliebigen Erfindungen gebaut. Sie beginnt bei Pfaden, die in der tatsächlichen Geschichte existierten:

- GPGPU- und Stream-Programmierung vor CUDA,
- OpenCL als offener heterogener Standard,
- CPU- und HPC-Cluster,
- verteilte Systeme wie DistBelief,
- FPGAs und anwendungsspezifische Beschleuniger,
- neuromorphe Systeme,
- Modellkompression, Sparsity und Quantisierung,
- Auto-Tuning und hardwarebewusste Compiler.

Die Quellen belegen, dass diese Pfade real waren. Sie belegen nicht, dass einer davon ohne CUDA gewonnen hätte.

Dieser Unterschied wird in jedem Szenario gewahrt:

> *Historischer Anker: Der Pfad existierte. Kontrafaktische Annahme: Er hätte unter verändertem Selektionsdruck mehr Aufmerksamkeit erhalten können. Offene Folge: Ob er dadurch leistungsfähiger, offener oder dominanter geworden wäre, bleibt unentschieden.*

Keine einzelne Ersatzgeschichte

Der Fehler vieler Gegengeschichten liegt darin, dass sie den realen Sieger durch einen alternativen Sieger ersetzen.

Ohne CUDA hätte demnach OpenCL gewonnen. Oder die TPU. Oder ein globaler CPU-Cluster. Oder neuromorphe Hardware.

Eine so saubere Substitution ist unwahrscheinlich. Der reale CUDA-Pfad gewann gerade deshalb an Macht, weil er viele Schichten integrierte. Fehlt dieser Integrationspunkt, kann die wahrscheinlichste Folge Fragmentierung sein.

Dieses Buch entwickelt deshalb mehrere konkurrierende Welten:

1. eine offene, compilerzentrierte Hardwarepluralität;
2. ein stärker konzentriertes Oligopol proprietärer Spezialchips;
3. eine langsamere, fragmentierte Entwicklung ohne klaren Standard.

Keine Welt wird als Wahrheit ausgewählt. Sie dienen als Belastungstest unterschiedlicher Kausalannahmen.

Was bleibt konstant?

Ein Gegenfaktum benötigt nicht nur eine veränderte Bedingung. Es benötigt Annahmen darüber, was zunächst konstant bleibt.

Für die ersten Schritte bleiben bestehen:

- der Markt für leistungsfähige Grafikchips,
- das Wachstum digitaler Daten,
- Fortschritte bei CPUs, Netzwerken und Halbleiterfertigung,
- Forschung an mehrschichtigen neuronalen Netzen,
- industrielle Nachfrage nach Sprach-, Bild- und Empfehlungssystemen,
- sowie die Existenz großer, kapitalstarker Plattformunternehmen.

Mit zunehmender zeitlicher Entfernung werden diese Annahmen unsicherer. Eine andere Softwareplattform kann Chipinvestitionen verändern. Andere Chips verändern attraktive Algorithmen. Andere Algorithmen verändern Benchmarks und Produkte.

Das Gegenfaktum wird daher nicht bis ins Jahr 2026 als detaillierte Chronik fortgeschrieben. Es öffnet einen Verzweigungsraum und verfolgt robuste Richtungen, keine fiktiven Quartalszahlen.

Drei Ebenen der Aussage

Um Spekulation und Befund nicht zu vermischen, verwendet der Text drei sprachliche Ebenen.

Belegt bezeichnet Aussagen, die eine Quelle direkt trägt: CUDA wurde 2006 eingeführt; OpenCL 1.0 erschien 2008; AlexNet wurde auf zwei GTX-580-GPUs trainiert; DistBelief skalierte neuronale Netze über sehr große CPU-Cluster.

Plausibel bezeichnet eine kausale Synthese aus mehreren belegten Eigenschaften: Ohne CUDA wäre die Reibung allgemeiner GPU-Programmierung wahrscheinlich höher geblieben; Compiler und Auto-Tuning hätten in einer heterogenen Landschaft vermutlich größere Bedeutung erhalten.

Prüfbar bezeichnet die Rückübersetzung in ein heutiges Experiment: Ein hardwarebewusstes Kostenmodell kann gegen feste Heuristiken getestet werden; ein kommunikationsarmes Verfahren kann auf einem bandbreitenbegrenzten Cluster gegen synchrone Baselines antreten.

Nur die dritte Ebene kann Project Ant Colony selbst empirisch entscheiden.

Der Gegenfaktums-Test

Jede alternative Behauptung muss fünf Fragen bestehen:

1. **Minimalität:** Wurde nur der definierte CUDA-Pfad entfernt?
2. **Anker:** Existierte die angenommene Alternative tatsächlich?
3. **Widerstand:** Bleiben physische, ökonomische und institutionelle Kosten sichtbar?
4. **Konkurrenz:** Gibt es mindestens ein plausibles Gegenszenario?
5. **Rückkehr:** Lässt sich aus der Annahme eine heutige, falsifizierbare Frage ableiten?

Scheitert eine Behauptung an diesen Fragen, bleibt sie literarische Spekulation. Das ist nicht wertlos, gehört aber nicht zum argumentativen Kern dieses Buches.

Wozu die Abzweigung dient

Das Gegenfaktum soll nicht beweisen, dass der reale Pfad falsch war.

CUDA beseitigte reale Reibung. Es machte Forschung schneller, reproduzierbarer und für viele Gruppen zugänglich. Die alternative Geschichte muss diese Leistung nicht verkleinern.

Sie fragt etwas anderes:

> *Welche technischen Freiheitsgrade wurden durch den Erfolg des integrierten Pfades so unwichtig, dass sie aus dem üblichen Entwurf verschwanden?*

Diese Frage führt unmittelbar zu Kommunikation, Heterogenität, Sparsity, lokalen Lernschritten, spezialisierten Modellen und compilerzentriertem Co-Design.

Nun kann der definierte Eingriff vorgenommen werden.

KAPITEL 12

Wenn es CUDA nie gegeben hätte

Der übersehene Evolutionszweig

Ohne den CUDA-Highway konkurrieren CPU-Cluster, offene Stacks, Spezialchips und algorithmische Sparsamkeit.

CUDA erfand weder parallele Berechnung noch neuronale Netze. CUDA verband eine weit verbreitete Klasse leistungsfähiger Grafikprozessoren mit einem zugänglichen Programmiermodell und ließ daraus über Jahre einen integrierten Stack wachsen.

Was geschieht, wenn genau diese Verbindung aus der Geschichte entfernt wird?

Die kurze Antwort lautet: KI wäre nicht stehen geblieben.

Die ehrliche Antwort lautet: Sie wäre wahrscheinlich langsamer, fragmentierter und in anderer Weise konzentriert gewachsen. Welche Linie dominant geworden wäre, lässt sich nicht wissen. Aber die realen Nebenpfade zeigen, welche Kräfte um den freien Raum konkurriert hätten.

Vor CUDA

Allgemeine Berechnung auf Grafikprozessoren existierte vor CUDA. Mit programmierbaren Grafikpipelines ließen sich nichtgrafische Aufgaben auf GPUs abbilden, häufig jedoch nur durch Übersetzung in Grafikbegriffe.

Brook for GPUs zeigte 2004 eine C-nahe Stream-Abstraktion. Compiler und Runtime behandelten die GPU als datenparallelen Coprozessor.[1] Das Projekt belegte zweierlei: Die Rechenressource war attraktiv, und ihre Programmierbarkeit war ein eigener Engpass.

CUDA entfernte einen großen Teil dieser Reibung. Ohne CUDA wäre die Nachfrage nach paralleler Rechenleistung dennoch geblieben. Andere Abstraktionen hätten mehr Gewicht tragen müssen.

Pfad eins: CPU-Cluster und HPC

CPUs besitzen geringeren Durchsatz für viele dichte Tensoroperationen als spezialisierte Beschleuniger. Dafür bieten sie flexible Kontrollflüsse, große Adressräume und ein ausgereiftes Ökosystem aus Compilern, Vektorbibliotheken, Threads und MPI.

DistBelief zeigte 2012 einen realen Skalierungspfad über Tausende Maschinen und Zehntausende CPU-Kerne. Das System kombinierte Daten- und Modellparallelität mit verteilten Optimierungsverfahren.[2]

Ohne CUDA wäre diese Linie vermutlich wichtiger geworden. Das hätte nicht einfach „heutige Modelle auf langsameren Prozessoren“ bedeutet. Ein CPU-Cluster zwingt andere Fragen in den Vordergrund:

- Wie viel lokale Arbeit geschieht vor der nächsten Synchronisation?
- Welche Worker sollen überhaupt teilnehmen?
- Wie werden langsame oder ausfallende Knoten behandelt?
- Welche Zustände müssen global konsistent sein?
- Wann kostet Kommunikation mehr als zusätzliche lokale Berechnung?

Asynchronität, Fehlertoleranz und Datenplatzierung wären wahrscheinlich früher zu Kerneigenschaften des Lernalgorithmus geworden.

Der Preis wäre beträchtlich: viele Maschinen, hoher Betriebsaufwand, Netzwerkkosten und schwierigere Reproduzierbarkeit.

Pfad zwei: ein offener heterogener Stack

Die Khronos Group veröffentlichte OpenCL 1.0 im Dezember 2008 als offenen, lizenzfreien Standard für parallele Programmierung über CPUs, GPUs und andere Prozessoren.[3]

Ohne CUDA hätte OpenCL oder eine verwandte Zwischendarstellung mehr institutionellen Raum erhalten können. Ein gemeinsamer Standard hätte Hardwarehersteller unter einer Programmierschnittstelle konkurrieren lassen.

Doch Portabilität ist nicht Leistung. Unterschiedliche Prozessoren besitzen unterschiedliche Speicherhierarchien, Vektorbreiten, Synchronisationskosten und Treiber. Derselbe korrekte Kernel kann auf mehreren Geräten sehr verschieden schnell laufen.

Unter diesen Bedingungen wäre eine zusätzliche Schicht strategisch geworden: Compiler, Kostenmodelle und Auto-Tuner, die eine abstrakte Operation an den konkreten Körper binden.

TVM demonstrierte später genau diese Richtung. Graphoptimierung, Operatorgenerierung und hardwarebezogene Suche verbinden ML-Modelle mit CPUs, GPUs und spezialisierten Zielsystemen.[4] Ansor erweiterte die automatische Suche in großen Tensorprogramm-Räumen.

Eine solche Welt hätte mehr Hardwarevielfalt erlaubt. Sie hätte zugleich mehr Treiberprobleme, schwankende Leistung und höhere Integrationskosten erzeugt.

Pfad drei: früheres Spezial-Silizium

Fehlt ein allgemein zugänglicher GPU-Stack, steigt der Anreiz, wichtige Operationen direkt in spezielle Hardware zu gießen.

Googles erste Tensor Processing Unit war ein anwendungsspezifischer Beschleuniger für neuronale Inferenz in Rechenzentren.[5] Microsofts Project Brainwave nutzte FPGAs für verteilte Inferenz.[6] IBM TrueNorth verfolgte eine ereignisgetriebene neuromorphe Architektur, weit entfernt von einem klassischen dichten Tensorpfad.[7]

Ohne CUDA hätten FPGAs, ASICs, neuromorphe Chips und speichernahe Recheneinheiten früher oder stärker konkurrieren können.

Dieser Pfad besitzt zwei entgegengesetzte politische Möglichkeiten.

Offene Compiler und mehrere Anbieter könnten eine pluralistische Beschleunigerlandschaft schaffen. Ebenso plausibel ist ein Oligopol großer Unternehmen, die sich eigene Chips und Compilerteams leisten können. In diesem Fall wäre Spitzen-KI noch geschlossener geworden.

CUDA war proprietär, aber programmierbare Consumer-GPUs gaben vielen Forschungsgruppen Zugang zu erheblicher Parallelleistung. Seine Abwesenheit wäre daher nicht automatisch demokratischer gewesen.

Pfad vier: algorithmische Sparsamkeit

Wenn Rechenzeit teuer und fragmentiert bleibt, verändert sich der Selektionsdruck auf Modelle.

Ein leicht zugänglicher Beschleuniger belohnt Verfahren, die zusätzliche Tensorberechnung zuverlässig in bessere Ergebnisse umsetzen. Eine knappe Landschaft belohnt stärker, was Speicher, Kommunikation und aktive Recheneinheiten vermeidet.

Dadurch hätten folgende Richtungen früher zum Ausgangspunkt werden können:

- Sparsity und bedingte Aktivierung,
- Quantisierung und niedrige Präzision,
- Pruning und kompakte Repräsentationen,
- kleine Spezialisten und modulare Systeme,
- externe Speicher und Retrieval,
- Wiederverwendung von Zwischenergebnissen,
- kommunikationsarme oder asynchrone Optimierung.

Deep Compression zeigte später, wie stark Pruning, trainierte Quantisierung und Codierung den Speicherbedarf konkreter Netze reduzieren konnten.[8] Die Arbeit beweist nicht, dass eine rechenarme Welt automatisch effizientere Modelle hervorgebracht

hätte. Sie belegt, dass Modellform und Ressourcenbedarf nicht fest miteinander verbunden sind.

Die alternative Entwicklungsfrage lautet daher:

> *Nicht: Wie verkleinern wir ein bereits erfolgreiches großes Modell? Sondern: Welche Lernform entsteht, wenn Ressourcen von Beginn an Teil der Zielfunktion sind?*

Die Physik bleibt, die Gewichtung ändert sich

Alle vier Pfade müssen dieselben fundamentalen Kosten tragen. Berechnung, Speicher und Kommunikation verschwinden nicht.

Was sich ändert, ist ihre Sichtbarkeit.

Im CPU-Cluster wird Kommunikation zum Algorithmus. Im offenen Stack wird Hardwarebindung zum Compilerproblem. Im ASIC-Pfad wird Chipzugang zur institutionellen Grenze. Im sparsamen Pfad wird die Zahl tatsächlich aktiver Operationen zum Entwurfsziel.

CUDA löste viele dieser Spannungen nicht endgültig. Es bündelte sie in einem Stack, der so produktiv war, dass seine bevorzugte Gewichtung zum Standard wurde.

Die Brücke

Das Gegenfaktum liefert noch keine alternative Geschichte. Es liefert einen Satz plausibler Selektionsdrücke.

Diese Drücke können zu sehr verschiedenen Gesellschaften und Forschungsordnungen führen. Eine offene heterogene Welt ist ebenso möglich wie ein stärker proprietäres ASIC-Oligopol. Eine fragmentierte Entwicklung kann mehr Vielfalt bewahren und zugleich Fortschritt verlangsamen.

Darum braucht das Buch mehr als einen Ersatzpfad.

Es braucht drei mögliche Welten.

Quellenhinweise

1 Ian Buck et al., „Brook for GPUs: Stream Computing on Graphics Hardware“, *ACM Transactions on Graphics* 23, Nr. 3 (2004), 777-786, DOI: 10.1145/1015706.1015800.

2 Jeffrey Dean et al., „Large Scale Distributed Deep Networks“, *Advances in Neural Information Processing Systems 25* (2012), 1223-1231.

3 Khronos Group, „OpenCL 1.0 Released“, 8. Dezember 2008.

4 Tianqi Chen et al., „TVM: An Automated End-to-End Optimizing Compiler for Deep Learning“, *13th USENIX Symposium on Operating Systems Design and Implementation* (2018), 578-594.

5 Norman P. Jouppi et al., „In-Datacenter Performance Analysis of a Tensor Processing Unit“, *44th International Symposium on Computer Architecture* (2017), 1-12.

6 Eric Chung et al., „Serving DNNs in Real Time at Datacenter Scale with Project Brainwave“, *IEEE Micro* 38, Nr. 2 (2018), 8-20.

7 Filipp Akopyan et al., „TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip“, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34, Nr. 10 (2015), 1537-1557.

8 Song Han, Huizi Mao und William J. Dally, „Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding“, *International Conference on Learning Representations* (2016), arXiv:1510.00149.

KAPITEL 13

Drei mögliche Welten

Gegen die bequeme Utopie

Offene Pluralität, ASIC-Oligopol und langsame Fragmentierung verhindern die bequeme Wunschgeschichte.

Eine Welt ohne CUDA lässt sich leicht romantisieren.

Offene Standards hätten gewonnen. Alte Rechner wären länger nützlich geblieben. Modelle wären kleiner und sparsamer. Forschung wäre weniger von einzelnen Unternehmen abhängig.

Diese Welt ist möglich. Sie ist nicht die einzige.

Dass ein dominanter proprietärer Stack fehlt, kann Wettbewerb öffnen. Es kann ebenso dazu führen, dass nur Unternehmen mit eigener Chipentwicklung die Reibung überwinden. Fragmentierung kann Vielfalt schützen. Sie kann auch verhindern, dass Forschungsergebnisse reproduzierbar werden.

Die alternative Evolution wird daher in drei Welten erzählt. Sie sind keine vollständigen Zukünfte, sondern konsistente Bündel von Selektionskräften.

Welt eins: offene Pluralität

In der ersten Welt wird keine einzelne Anbieterplattform zum unumstrittenen Zentrum. OpenCL, LLVM-basierte Zwischendarstellungen oder verwandte Abstraktionen bilden einen gemeinsamen, aber unvollkommenen Untergrund.

CPUs, GPUs verschiedener Hersteller, FPGAs und Spezialchips bleiben gleichzeitig relevant. Modelle werden nicht direkt für ein Gerät geschrieben. Compiler und Auto-Tuner erzeugen gerätespezifische Pläne.

Die zentrale technische Institution dieser Welt ist nicht der Beschleunigerhersteller. Es ist die Übersetzungsschicht.

Forschung investiert stärker in:

- portable Rechengraphen,
- gelernte Kostenmodelle,
- automatische Operatorerzeugung,

- hardwarebezogene Zwischendarstellungen,
- und Benchmarks über mehrere Geräteklassen.

Die Modellseite reagiert. Regelmäßige dichte Operationen bleiben attraktiv, aber Portabilität, Sparsity und Modularität erhalten früher einen eigenen Wert. Ein Modell, das nur auf einem Zielsystem schnell ist, gilt weniger selbstverständlich als allgemeiner Fortschritt.

Der Gewinn

Kein einzelner Hersteller kontrolliert die gesamte Entwicklungskette. Neue Hardware kann durch einen gemeinsamen Compilerstack zugänglich werden. Forschung auf unterschiedlichen Geräten bleibt vergleichbarer.

Algorithmische Vielfalt erhält mehr Chancen, weil die Übersetzung zwischen Modell und Maschine als normales Forschungsproblem gilt.

Der Preis

Offene Spezifikationen garantieren keine guten Implementierungen.

Treiber unterscheiden sich. Fehler treten nur auf einzelnen Geräten auf. Performance-Portabilität bleibt schwierig. Forschungsteams verbringen mehr Zeit mit Integration. Ergebnisse können auf nominell kompatibler Hardware stark abweichen.

Die offene Welt ist pluralistischer und anstrengender.

Welt zwei: das ASIC-Oligopol

In der zweiten Welt bleibt allgemeine GPU-Programmierung mühsam. Plattformunternehmen reagieren nicht mit Offenheit, sondern mit vertikaler Integration.

Sie entwickeln eigene Beschleuniger, Compiler, Netzwerke und Rechenzentrumsarchitekturen. Die Chips sind hervorragend auf ausgewählte neuronale Operationen abgestimmt. Externe Nutzer greifen über Cloudangebote oder APIs zu.

Die zentrale Institution ist das integrierte Unternehmen.

Universitäten können Modelle auf gemieteten Ressourcen trainieren, besitzen aber weder den vollständigen Stack noch dieselbe Diagnosetiefe. Der Zugang zu Spitzenleistung wird dienstförmig: Die Rechenkapazität ist erreichbar, aber nicht unabhängig reproduzierbar.

Der Gewinn

Spezialchips können für definierte Lasten hohe Leistung und Energieeffizienz erreichen. Hardware, Compiler und Modellteams arbeiten eng zusammen. Rechenzentren werden als Gesamtsystem optimiert.

Der Preis

Nur wenige Organisationen können die Entwicklungskosten tragen. Ihr Hardwarepfad prägt, welche Modelle wirtschaftlich sind. Technisches Wissen, Trainingsdaten und Fertigungskapazität konzentrieren sich.

Ohne den programmierbaren Consumer-GPU-Pfad könnte die Lücke zwischen akademischer Forschung und industrieller Spitze größer werden.

Diese Welt ist möglicherweise effizienter - und geschlossener.

Welt drei: langsame Fragmentierung

In der dritten Welt gewinnt niemand.

CPU-Cluster, verschiedene GPU-Sprachen, FPGAs, akademische Stream-Abstraktionen und frühe Spezialchips konkurrieren. Jede Gemeinschaft entwickelt eigene Werkzeuge. Portierung ist teuer; Ergebnisse sind schwer vergleichbar.

Deep Learning wächst dennoch. Seine Experimente dauern länger, und große Trainingsläufe bleiben seltener. Symbolische, probabilistische und hybride Ansätze behalten mehr institutionellen Raum, nicht zwingend weil sie überlegen sind, sondern weil kein Beschleunigerpfad die Konkurrenz so schnell überrollt.

Die zentrale Institution dieser Welt ist das lokale Labor.

Forschung wird stärker durch vorhandene Maschinen geprägt. Ein Institut mit einem CPU-Cluster verfolgt andere Modelle als ein Unternehmen mit FPGAs. Globale Benchmarks entwickeln weniger normierende Kraft, weil Implementierungen schwer übertragbar sind.

Der Gewinn

Der Suchraum bleibt länger offen. Verschiedene Modellformen und Hardwareannahmen überleben. Lokales Systemwissen besitzt hohen Wert.

Der Preis

Jede Idee bezahlt erneut für ihre Infrastruktur. Fehler werden wiederholt. Kleine Gruppen können weniger auf stabilen Bibliotheken aufbauen. Fortschritt wird langsamer, und Reproduzierbarkeit leidet.

Vielfalt entsteht hier nicht aus Freiheit, sondern aus fehlender Standardisierung.

Die Welten vermischen sich

Technikgeschichte verläuft nicht global einheitlich.

In einer Region kann offene Compilerforschung dominieren, während ein Plattformunternehmen parallel ein proprietäres ASIC-Ökosystem aufbaut. Akademische Labore können fragmentiert bleiben, obwohl Cloudanbieter hochintegrierte Systeme betreiben. Ein offener Standard kann die Programmierschnittstelle vereinheitlichen, während Fertigung und Trainingskapazität konzentriert bleiben.

Die drei Welten sind daher keine voneinander isolierten Planeten. Sie sind Kräftefelder:

Integrationszentrum	
Offene Pluralität	Compiler und Standards
ASIC-Oligopol	vertikales Unternehmen
Langsame Fragmentierung	lokales Labor

Hardwarevielfalt	
Offene Pluralität	hoch und adressierbar
ASIC-Oligopol	gering an der Spitze
Langsame Fragmentierung	hoch, aber inkompatibel

Reproduzierbarkeit	
Offene Pluralität	mittel
ASIC-Oligopol	intern hoch, extern begrenzt
Langsame Fragmentierung	niedrig

Zugang	
Offene Pluralität	breit, mit Integrationskosten
ASIC-Oligopol	dienstförmig und konzentriert
Langsame Fragmentierung	lokal und ungleich

Modellselektionsdruck	
Offene Pluralität	portabel und adaptiv
ASIC-Oligopol	chipnah und skaliert
Langsame Fragmentierung	nischen- und hardwareabhängig

Die Tabelle bewertet keine Welt insgesamt als besser. Jede verschiebt Kosten.

Wo läge MOS?

Das Model OS - kurz MOS - hätte in jeder Welt eine andere Rolle.

In der offenen Pluralität wäre MOS eine naheliegende Übersetzungsschicht. Es würde den Systemzustand messen und Modelle, Operatoren oder Jobs auf unterschiedliche Geräte binden.

Im ASIC-Oligopol könnte MOS als interner Dienst eines Plattformunternehmens entstehen: leistungsfähig, aber nicht allgemein verfügbar.

In der fragmentierten Welt wäre MOS am wertvollsten und am schwersten zu bauen. Es müsste lokale Besonderheiten verstehen, für die es wenig gemeinsame Trainingsdaten und instabile Schnittstellen gibt.

Diese Beobachtung schärft die Forschungslücke. Der Nutzen eines Entscheidungsmodells steigt mit Heterogenität; zugleich sinkt die Menge vergleichbarer Erfahrung, aus der es lernen kann.

Die gesellschaftliche Auswahl

Technik allein entscheidet nicht, welche Welt wahrscheinlicher wird.

Beschaffungsregeln, offene Forschungsförderung, Exportkontrollen, Cloudökonomie, Energiepreise und Ausbildung wirken mit. Ein offener Standard braucht Pflege. Ein ASIC-Oligopol braucht Kapital und Fertigungszugang. Fragmentierung bleibt bestehen, wenn niemand die Kosten gemeinsamer Infrastruktur trägt.

Damit kehrt das Buch zu Kapitel 2 zurück. Randbedingungen sind keine Kulisse. Auch im Gegenfaktum wählen Institutionen mit.

Der gemeinsame Nenner

So verschieden die drei Welten sind, teilen sie eine Eigenschaft: Der Algorithmus kann nicht so selbstverständlich als feste Größe behandelt werden wie im reifen CUDA-Stack.

In der offenen Welt muss er portierbar oder automatisch übersetzbar sein. Im ASIC-Oligopol muss er zum Spezialchip passen. In der fragmentierten Welt muss er lokale Grenzen akzeptieren.

Jede Welt erzeugt daher stärkeren Druck auf Co-Design.

Die nächste Frage lautet nicht mehr nur, welche Infrastruktur entstanden wäre. Sie lautet:

> *Welche Form von KI hätte unter diesen Selektionsbedingungen gute Überlebenschancen gehabt?*

KAPITEL 14

Welche KI hätte diese Welt hervorgebracht?

Nicht dieselben Modelle auf langsameren Maschinen

Andere Ressourcenpreise begünstigen Spezialisten, Modularität, externe Speicher und compilerzentriertes Co-Design.

Die schwächste Vorstellung einer Welt ohne CUDA kopiert die Gegenwart und entfernt nur Geschwindigkeit.

Heutige Transformer würden auf CPU-Clustern trainiert. Alles dauerte länger. Große Sprachmodelle kämen einige Jahre später, sähen aber im Wesentlichen gleich aus.

Dieses Szenario ist möglich. Es unterschätzt jedoch Co-Design.

Wenn Rechenzeit, Speicher und Kommunikation dauerhaft andere Preise besitzen, werden nicht nur Implementierungen verändert. Architekturen und Lernverfahren reagieren. Die alternative KI wäre deshalb wahrscheinlich kein langsameres Abbild der realen. Sie würde andere Eigenschaften früh belohnen.

Kleinere Spezialisten

Ein generalistisches Foundation Model amortisiert ein sehr teures Training über viele Aufgaben. Diese Logik ist stark, wenn große Trainingsläufe finanzierbar und die resultierenden Modelle breit einsetzbar sind.

In einer fragmentierten Rechenwelt kann die Rechnung anders ausfallen. Kleine Sprach-, Code-, Wahrnehmungs- und Planungsmodelle werden lokal trainiert oder angepasst. Eine Orchestrierung wählt den passenden Spezialisten.

Der Vorteil liegt nicht nur in weniger Parametern. Spezialisten können:

- engere Datenverteilungen nutzen,
- kleinere Qualitätskriterien erfüllen,
- auf bestimmte Hardwareklassen zugeschnitten werden,
- getrennt aktualisiert und geprüft werden,
- und bei Nichtgebrauch vollständig inaktiv bleiben.

Der Preis ist Koordination. Das System muss erkennen, welcher Spezialist zuständig ist, wie Ergebnisse zusammengeführt werden und was geschieht, wenn keiner genügt.

In dieser Welt wird Routing zu einer Kernkompetenz der KI.

Modularität und bedingte Aktivierung

Ein dichtes Modell aktiviert für eine Anfrage einen großen Teil seiner Parameter. Diese Regelmäßigkeit passt hervorragend zu großen Matrixmultiplikationen.

Unter knapperer Berechnung wird Kapazität attraktiver, die nicht immer aktiv sein muss. Mixture-of-Experts-Architekturen zeigen im realen Verlauf, dass Modelle sehr viele Parameter besitzen können, während pro Eingabe nur ausgewählte Experten berechnet werden.[1]

Im Gegenfaktum könnte bedingte Aktivierung früher zur Grundform werden:

- ein Router wählt wenige Module,
- lokale Experten arbeiten auf begrenzten Ausschnitten,
- Speicher wird von Rechenaktivität entkoppelt,
- und neue Fähigkeiten können als zusätzliche Module entstehen.

Diese Architektur spart nicht automatisch Ressourcen. Routing, Lastverteilung und Expertenkommunikation erzeugen eigene Kosten. Auf einem langsamen Netzwerk kann ein schlecht platzierter Experte teurer sein als ein kleineres dichtes Modell.

Gerade deshalb würde Modellarchitektur untrennbar mit Platzierung werden.

Gedächtnis außerhalb der Gewichte

Große Modelle speichern einen Teil ihrer Fähigkeiten implizit in Parametern. Externe Speicher- und Retrievalsysteme bieten einen anderen Weg: Das Modell erzeugt Suchanfragen, ruft relevante Dokumente oder Zustände ab und verarbeitet sie im Kontext.

Retrieval-Augmented Generation demonstrierte im realen Forschungsweg die Verbindung eines parametrischen Sequenzmodells mit einem nichtparametrischen Dokumentenspeicher.[2]

In einer rechenärmeren Evolution hätte diese Trennung früher Gewicht erhalten:

- Lernen erzeugt nicht für jede Information neue Parameter.
- Aktualisierung kann durch Änderung des Speichers erfolgen.
- lokales Wissen bleibt lokal speicherbar.
- verschiedene Geräte können Rechnen und Gedächtnis übernehmen.

Die Grenze verschiebt sich von Parameterzahl zu Zugriff. Suchlatenz, Indexqualität, Datenherkunft und Kontextbudget werden Teil der Systemintelligenz.

Kommunikation als Lernvariable

Der reale Beschleunigerstack arbeitet intensiv daran, Kommunikation unsichtbar zu machen: schnelle Verbindungen, optimierte Kollektive und überlappte Datenbewegung.

In einem Commodity-Cluster bleibt sie sichtbar. Eine Lernmethode muss entscheiden, wie häufig Zustände synchronisiert werden und welche Information überhaupt übertragen wird.

Local-SGD-Verfahren führen mehrere lokale Updates aus, bevor Modelle zusammengeführt werden. Kommunikationskompression überträgt reduzierte oder ausgewählte Gradienten. Asynchrone Verfahren erlauben unterschiedlichen Workern, mit zeitlich versetzten Zuständen zu arbeiten.

Diese Methoden existieren in der realen Forschung.[3] Die alternative Annahme lautet nur, dass sie nicht als Speziallösung für eine schwierige Verteilung, sondern als normaler Ausgangspunkt des Lernens behandelt worden wären.

Dann wäre die Frage „Wann synchronisieren wir?“ ebenso grundlegend wie die Wahl des Optimierers.

Der Compiler als Mitautor

In einem homogenen Stack kann ein Modell weitgehend unabhängig von der konkreten Gerätegeneration beschrieben werden. Bibliotheken übernehmen den Rest.

In einer heterogenen Welt reicht diese Trennung nicht. Ein Compiler müsste nicht nur Operatoren übersetzen. Er müsste Entscheidungen über Rechenstruktur treffen:

- Welche Präzision ist auf welchem Knoten sinnvoll?
- Welche Operationen werden fusioniert?
- Welche Zwischenergebnisse werden gespeichert oder neu berechnet?
- Welche Module werden repliziert?
- Welche Daten bleiben lokal?
- Wann wird ein langsamer Worker ausgeschlossen?

TVM und Ansor zeigen reale Teile dieses Pfades: hardwarebezogene Operatorerzeugung, Suche und gelernte Kostenmodelle.[4]

Die alternative KI wäre damit compilerzentrierter. Das Modell beschreibt eine Absicht; die konkrete Berechnung entsteht in Auseinandersetzung mit dem gemessenen Körper.

Wären große Sprachmodelle entstanden?

Wahrscheinlich ja - aber nicht zwingend zur gleichen Zeit oder in derselben Form.

Die wichtigsten Voraussetzungen bleiben bestehen: digitale Textmengen, Gradientenverfahren, Attention, Rechenzentren und wirtschaftliches Interesse. CUDA ist keine notwendige mathematische Bedingung eines Sprachmodells.

Fünf Verläufe bleiben plausibel:

1. **Spätere LLMs:** Die gleiche grobe Architektur skaliert langsamer.
2. **Modulare LLMs:** Experten, Retrieval und hierarchisches Routing werden früher zentral.
3. **Stärkere Spezialisierung:** Viele Domänenmodelle ersetzen einen Teil der Generalistenlogik.
4. **Noch größere Konzentration:** Nur Unternehmen mit eigenen ASICs können große Modelle trainieren.
5. **Längere Paradigmenkonkurrenz:** Symbolische, probabilistische und hybride Systeme behalten mehr Forschungsanteil.

Diese Verläufe schließen einander nicht aus. Ein proprietärer Generalist kann neben offenen lokalen Spezialisten existieren.

Eine andere Definition von Größe

Im realen Narrativ wird Größe häufig durch Parameter, Daten und Trainings-FLOPs beschrieben.

Eine alternative Evolution könnte Systemgröße anders messen:

- Zahl verfügbarer Spezialisten,
- Umfang eines externen Gedächtnisses,
- Zahl nutzbarer Geräteklassen,
- Zeit, über die lokale Erfahrung gesammelt wird,
- oder Vielfalt der ausführbaren Pläne.

Ein System kann groß sein, ohne bei jeder Anfrage groß zu rechnen.

Diese Definition passt zur Ameisenkolonie. Ihre Leistungsfähigkeit liegt nicht in einer riesigen Ameise, sondern in vielen begrenzten Einheiten, Kommunikationsspuren und einer Ordnung, die sich an die Umgebung anpasst.

Die Metapher ist keine Architektur. Sie macht jedoch eine veränderte Entwurfsintuition sichtbar.

Der übersehene Freiheitsgrad

Der dominante Stack behandelt den Algorithmus häufig als gegeben und sucht die beste Ausführung auf geeigneter Hardware.

Die alternative Evolution behandelt beides als veränderbar:

> *Algorithmus = Funktion aus Aufgabe, Qualitätsziel und gemessenem Systemzustand.*

Das ist eine stärkere Behauptung als Auto-Tuning. Nicht nur Blockgrößen und Kernelauswahl können wechseln. Auch Synchronisationsschema, aktive Knoten, Modellmodule, Präzision und Zeitpunkt werden zu Entscheidungen.

Damit endet das Gegenfaktum.

Die Frage ist nun nicht mehr, welche KI eine andere Vergangenheit hervorgebracht hätte. Die Frage lautet, ob sich diese Logik heute auf einer realen heterogenen Plattform prüfen lässt.

Der Algorithmus selbst wird zur Variablen.

Quellenhinweise

1 William Fedus, Barret Zoph und Noam Shazeer, „Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity“, *Journal of Machine Learning Research* 23, Nr. 120 (2022), 1-39.

2 Patrick Lewis et al., „Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks“, *Advances in Neural Information Processing Systems 33* (2020), 9459-9474.

3 Sebastian U. Stich, „Local SGD Converges Fast and Communicates Little“, *International Conference on Learning Representations* (2019).

4 Tianqi Chen et al., „TVM: An Automated End-to-End Optimizing Compiler for Deep Learning“, *13th USENIX Symposium on Operating Systems Design and Implementation* (2018), 578-594; Lianmin Zheng et al., „Ansor: Generating High-Performance Tensor Programs for Deep Learning“, *14th USENIX Symposium on Operating Systems Design and Implementation* (2020), 863-879.

ZWISCHENSPIEL

Die Softwarewelt, die nicht entstand

Vom Rechenkern bis zur Anwendung

Der alternative Evolutionsstrang reicht von portablen Rechenkernen über Betriebssysteme bis zu Anwendungen und ihren gesellschaftlichen Folgen.

Ein KI-Modell lebt nie allein.

Zwischen einer mathematischen Idee und einer Anwendung auf dem Bildschirm liegen viele Schichten. Ein Compiler übersetzt Operationen. Ein Rechenkern führt sie auf einem Gerät aus. Eine Laufzeitumgebung bewegt Daten. Ein Betriebssystem verteilt Speicher und Rechenzeit. Erst darauf entsteht eine Anwendung, die für ihre Nutzerinnen und Nutzer wie ein zusammenhängendes Produkt aussieht.

Wenn sich die Entwicklung der KI an einer Stelle verzweigt, verzweigen sich deshalb auch diese Schichten.

Die Frage nach einer Welt ohne CUDA betrifft nicht nur die Größe von Modellen. Sie betrifft auch, wie Programme gebaut, verteilt und gewartet worden wären. Sie betrifft, was ein Betriebssystem über seine Geräte wissen müsste. Und sie betrifft die alltägliche Erwartung, ob eine Anwendung lokal arbeitet, eine Cloud benötigt oder sich ihren Ausführungsort selbst sucht.

Damit erreicht die alternative Evolution den gewöhnlichen Computer.

Zwei Bedeutungen von Kernel

Das Wort *Kernel* besitzt in dieser Geschichte zwei Bedeutungen.

Ein Compute-Kernel ist ein kleines Programm für eine bestimmte Rechenoperation. Es multipliziert Matrizen, faltet Bilder, sortiert Werte oder verändert Gewichte. Viele solcher Kerne bilden den körperlichen Vollzug eines Modells. Sie entscheiden mit darüber, ob eine Idee nur korrekt oder auch schnell genug ist.

Der Kernel eines Betriebssystems erfüllt eine andere Aufgabe. Er verwaltet Prozesse, Speicher, Geräte und Rechte. Er entscheidet, welche Arbeit wann ausgeführt werden darf und welche Ressourcen ein Programm überhaupt sieht.

Im erfolgreichen Beschleunigerpfad wurden viele Compute-Kernels innerhalb eines dichten Hersteller-, Treiber- und Bibliotheksstacks zuverlässig verfügbar. Anwendungen mussten die Verschiedenheit der Geräte immer weniger selbst verstehen. Ein großer Teil der Komplexität verschwand hinter stabilen Schnittstellen.

Ohne diesen frühen Integrationspfad wäre die Komplexität nicht verschwunden. Sie hätte einen anderen Wohnort gebraucht.

Vielleicht wäre sie stärker in Compiler gewandert. Vielleicht in Betriebssysteme. Vielleicht hätten Anwendungen mehrere Varianten mitbringen müssen. Vielleicht hätte ein anderer Anbieter die Integration später auf proprietärem Silizium nachgeholt.

Das Gegenfaktum beginnt daher nicht mit einer bestimmten Ersatztechnik. Es beginnt mit einer offenen Zuständigkeitsfrage:

> *Welche Softwareschicht hätte die Vielfalt der Hardware zusammenhalten müssen?*

Übersetzung wird zur Hauptaufgabe

In einer offeneren Hardwarewelt kann ein Programm nicht selbstverständlich für ein einziges Ziel geschrieben werden.

Eine mögliche Antwort besteht aus Zwischensprachen. Der Quelltext beschreibt eine Berechnung zunächst unabhängig vom endgültigen Gerät. Später wird diese Beschreibung für eine konkrete CPU, GPU, FPGA oder einen Spezialbeschleuniger weiter übersetzt.

SPIR-V ist ein realer Anker für diese Idee. Die Khronos Group beschreibt es als standardisierte Zwischenform für parallele Berechnung und Grafik, die von verschiedenen Sprachwerkzeugen erzeugt und von unterschiedlichen Treibern verarbeitet werden kann.[1] SYCL verfolgt einen verwandten Weg auf der Ebene von C++: Programme können Geräte entdecken, Daten verwalten und Rechenarbeit auf verschiedene Beschleunigerklassen verteilen.[2]

Auch MLIR antwortet auf die Fragmentierung von Compilern und Zielsystemen. Es stellt mehrere Abstraktionsebenen bereit, damit eine Berechnung schrittweise von einer fachlichen Beschreibung zu hardwarenaher Ausführung übersetzt werden kann.[3]

Diese Projekte beweisen nicht, dass eine Welt ohne CUDA automatisch auf offene Zwischensprachen gesetzt hätte. Sie zeigen nur, dass der alternative Baustein real ist.

In einer solchen Welt wäre die wichtigste Software vielleicht nicht die beste handgeschriebene Implementierung für ein Gerät gewesen. Wichtiger hätte die Fähigkeit werden können, aus einer gemeinsamen Beschreibung mehrere gute Implementierungen zu erzeugen.

Der Kernel wäre dann weniger das endgültige Ziel als das Ergebnis einer Übersetzung.

Das Betriebssystem sieht keine gleichen Kerne mehr

Klassische Betriebssysteme können viele Unterschiede verbergen. Zwei Prozessorkerne erscheinen einer Anwendung ähnlich genug, um Arbeit zwischen ihnen zu verteilen. Bei heterogenen Geräten reicht diese Annahme nicht.

Ein schneller Kern, ein sparsamer Kern, ein Grafikprozessor und ein Spezialbeschleuniger besitzen nicht nur verschiedene Geschwindigkeiten. Sie besitzen andere Speicher, andere Zugriffswege und andere günstige Operationen.

Reale Betriebssystementwicklung zeigt, dass diese Unterschiede grundsätzlich sichtbar gemacht werden können. Linux besitzt Mechanismen für kapazitätsbewusste Planung auf ungleichen Prozessorkernen. Der Scheduler berücksichtigt, dass eine Aufgabe auf unterschiedlichen Kernen verschieden gut passt.[4] Die Dokumentation zur Heterogeneous Memory Management behandelt zudem die schwierige Verbindung zwischen gewöhnlichem Anwendungsspeicher und Gerätespeicher.[5]

Auch die Heterogeneous System Architecture formulierte standardisierte Schnittstellen, Warteschlangen und Speichermodelle für Systeme mit mehreren Arten von Recheneinheiten.[6]

Wieder gilt: Diese Technik ist kein verlorener Sieger. Sie ist ein Beleg dafür, dass ein Betriebssystem heterogene Rechenfähigkeit anders behandeln kann als durch einen vollständig getrennten Gerätestack.

In einer alternativen Evolution hätte das Betriebssystem vielleicht früher fragen müssen:

- Welche Art von Arbeit liegt vor?
- Welches Gerät kann sie überhaupt ausführen?
- Wo befinden sich die benötigten Daten?
- Ist Geschwindigkeit, Energie oder Verfügbarkeit wichtiger?
- Was geschieht, wenn ein Gerät während der Ausführung verschwindet?

Das Betriebssystem wäre dadurch nicht intelligent im menschlichen Sinn geworden. Aber es hätte ein reicheres Bild seines eigenen Körpers benötigt.

Anwendungen würden Fähigkeiten verlangen

Eine heutige Anwendung setzt häufig still voraus, welche Bibliothek, Cloud oder Geräteklasse ihre wichtigste Funktion ausführt.

In einer vielfältigeren Welt hätte sie anders gebaut sein können. Sie hätte nicht ein bestimmtes Gerät verlangt, sondern eine Fähigkeit beschrieben:

Ein Bild soll innerhalb einer Sekunde klassifiziert werden. Ein Text soll lokal bleiben. Eine Übersetzung darf langsam sein, wenn kein Netzwerk verfügbar ist. Eine medizinische Anwendung benötigt eine nachvollziehbare Ausführung. Ein Hintergrundprozess darf nur freie und sparsame Ressourcen verwenden.

Die Anwendung würde damit mehr über ihr Ziel und weniger über den exakten Ausführungsweg sagen.

Beim Start könnte die Laufzeitumgebung prüfen, welche Geräte vorhanden sind. Auf einem kleinen Rechner wählt sie ein kompaktes lokales Modell. Auf einer Workstation nutzt sie einen Beschleuniger. In einem Labor verteilt sie Arbeit über mehrere Knoten. Wenn keine gültige Variante existiert, lehnt sie die Funktion ab, statt heimlich ein unpassendes Verfahren zu verwenden.

Das klingt modern. Doch in der alternativen Geschichte wäre es vielleicht nicht die nachträgliche Anpassung an eine vielfältige Welt gewesen. Es hätte zur normalen Form von Software gehört.

Drei Softwarewelten

Auch diese Abzweigung führt nicht zu einer einzigen besseren Zukunft.

Die offene Welt

In einer offenen Hardwarepluralität investieren Gemeinschaften in portable Sprachen, Zwischendarstellungen, Testverfahren und freie Laufzeitumgebungen.

Anwendungen werden als Bündel mehrerer Varianten verteilt. Das lokale System wählt die passende. Hardware kann länger nutzbar bleiben, weil neue Software nicht ausschließlich an die jeweils neueste Gerätegeneration gebunden ist.

Der Preis ist Integrationsarbeit. Standards müssen gepflegt, Treiber getestet und Leistungsunterschiede erklärt werden. Offenheit beseitigt Komplexität nicht. Sie verteilt ihre Kosten.

Die vertikale Welt

In einem ASIC-Oligopol findet die Integration innerhalb weniger großer Unternehmen statt.

Kernel, Compiler, Betriebssystemdienste und Modelle werden gemeinsam für eigenes Silizium entwickelt. Anwendungen greifen über stabile Dienste auf diese Leistung zu. Für Nutzerinnen und Nutzer kann das sehr bequem sein.

Die sichtbare Fragmentierung sinkt, während die institutionelle Abhängigkeit steigt. Wer die Anwendung kontrolliert, kontrolliert nicht unbedingt das Modell, den Ausführungsplan oder die zugrunde liegende Hardware.

Ohne CUDA könnte die Konzentration daher größer und nicht kleiner geworden sein.

Die fragmentierte Welt

In einer langsamen, nicht integrierten Entwicklung entstehen viele lokale Softwareinseln.

Ein Labor besitzt gute Werkzeuge für seinen FPGA. Ein Hersteller pflegt eine Bibliothek für seinen Signalprozessor. Eine Universität betreibt einen CPU-Cluster mit eigenen Compilern. Anwendungen funktionieren dort gut, lassen sich aber nur schwer übertragen.

Diese Welt bewahrt Vielfalt, aber nicht automatisch Zugang. Wissen bleibt an Orte, Teams und Geräte gebunden. Kleine Organisationen können unabhängig arbeiten und zugleich von der Wartung ihrer eigenen Insel überfordert sein.

Software besitzt eine gesellschaftliche Form

Die Wahl einer Softwareschicht ist keine rein interne Ingenieursentscheidung.

Wenn eine Anwendung lokal laufen kann, verändert das Datenschutz, Verfügbarkeit und Eigentum an den entstehenden Daten. Wenn sie nur über eine Cloudschnittstelle funktioniert, verschieben sich Kontrolle und laufende Kosten. Wenn alte Geräte weiter unterstützt werden, verändert sich die Lebensdauer materieller Infrastruktur. Wenn jede Organisation eigene Compilerexpertinnen benötigt, entsteht eine neue Zugangshürde.

Keine dieser Folgen ist moralisch eindeutig.

Lokale Ausführung kann unabhängig machen und dennoch ineffizient sein. Eine Cloud kann Wissen konzentrieren und zugleich komplexe Technik breit zugänglich machen. Lange Hardwarelebensdauer kann Ressourcen sparen, aber unsichere oder energiehungrige Systeme konservieren. Offene Standards können Mitwirkung ermöglichen und an fehlender Pflege scheitern.

Gesellschaftliche Reflexion bedeutet deshalb nicht, aus einem technischen Nebenpfad sofort eine gerechtere Welt abzuleiten.

Sie bedeutet, sichtbar zu machen, dass ein technischer Standard auch Verantwortung verteilt:

- Wer muss die Komplexität beherrschen?
- Wer darf eine Anwendung prüfen?
- Wer kann sie ohne Erlaubnis weiter betreiben?
- Welche Geräte werden zu früh wertlos?
- Welche Kosten erscheinen im Preis, und welche verschwinden in der Infrastruktur?

Ein technischer Bruch beantwortet diese Fragen nicht. Er verändert aber, wer sie beantworten kann.

Von der Softwarewelt zu MOS

Damit schließt sich die Lücke zur eigentlichen Forschung.

In einem reifen, homogenen Stack darf eine Anwendung so tun, als sei der Algorithmus fest und die Ausführungsumgebung bloß ein Ziel. In einer heterogenen Softwarewelt trägt diese Annahme nicht.

Das System muss den gegenwärtigen Zustand kennen. Es muss Fähigkeiten statt Typenschilder lesen. Es muss zwischen mehreren gültigen Verfahren wählen und seine Entscheidung später erklären können.

Ein Model OS, kurz MOS, wird in dieser Perspektive verständlicher. Es ist kein allwissendes Betriebssystem und kein Ersatz für Kernel, Compiler oder Laufzeitumgebung.

Es ist ein begrenzter Entscheidungskopf zwischen ihnen.

Das trainierte Modell kann Vorschläge machen. Ein deterministischer Vertrag prüft, ob sie erlaubt sind. Die vorhandene Software führt nur gültige Pläne aus. Der Zustand, der Vorschlag und das Ergebnis bleiben sichtbar.

In unserer tatsächlichen Entwicklung wirkt diese Architektur ungewöhnlich. In einer Welt, deren Software von Anfang an mit Vielfalt leben musste, wäre sie vielleicht selbstverständlich erschienen.

Genau diese Möglichkeit führt nun zurück aus der alternativen Geschichte in die reale Architektur von Project Ant Colony.

Quellenhinweise

1 Khronos Group, „SPIR-V - The Industry Open Standard Intermediate Language for Parallel Compute and Graphics“, offizielle Übersicht.

2 Khronos Group, *SYCL 2020 Specification*, offene Spezifikation für heterogene Programmierung.

3 LLVM Project, „Multi-Level Intermediate Representation Overview“, offizielle Projektdokumentation.

4 Linux Kernel Documentation, „Capacity Aware Scheduling“.

5 Linux Kernel Documentation, „Heterogeneous Memory Management“.

6 Heterogeneous System Architecture Foundation, *HSA Platform System Architecture Specification*.

TEIL IV

Wenn der Algorithmus selbst variabel wird

MOS, Decision Compiler und ein lebender Systemzustand übersetzen die alternative Evolution in eine Architektur.

Kapitel 15 bis 18

KAPITEL 15

Nicht die Hardware anpassen, sondern das Verfahren

Die Umkehrung

Der Algorithmus wird als Funktion aus Aufgabe, Qualitätsziel und gemessenem Systemzustand behandelt.

Ein großer Teil moderner Systementwicklung folgt einer vertrauten Richtung.

Die Aufgabe und der Algorithmus stehen fest. Danach wird eine geeignete Hardware gewählt, beschafft oder gemietet. Compiler und Laufzeiten sorgen dafür, dass die Berechnung auf diesem Zielsystem effizient ausgeführt wird.

Project Ant Colony kehrt die Richtung um.

Die Hardware ist bereits vorhanden. Sie ist heterogen, älter, bandbreitenbegrenzt und nicht jederzeit vollständig verfügbar. Die Aufgabe besitzt ein Qualitätsziel. Veränderbar ist der Weg dazwischen.

> *Nicht die vorhandene Maschine wird an einen festen Algorithmus angepasst. Das Verfahren wird aus Aufgabe, Qualitätsgrenze und gemessenem Maschinenzustand gewählt.*

Diese Umkehrung ist kein Ersatz für gute Hardware. Sie ist eine andere Forschungsfrage.

Vom festen Algorithmus zur Verfahrensfamilie

Ein Algorithmus wird im alltäglichen Sprachgebrauch als eindeutige Folge von Schritten behandelt. Für ein verteiltes Lernsystem reicht dieses Bild nicht.

Schon eine bekannte Trainingsmethode besitzt Freiheitsgrade:

- Wie groß ist der globale Batch?
- Wie viele lokale Schritte führt ein Worker vor der Synchronisation aus?
- Werden Updates synchron oder begrenzt asynchron zusammengeführt?

- Werden Gradienten komprimiert?
- Welche Knoten nehmen teil?
- Wie wird mit langsamen Workern umgegangen?
- Wann wird neu geplant?

Diese Entscheidungen verändern nicht zwingend das mathematische Fernziel. Sie verändern jedoch den realen Lernweg, die Kommunikation, den Ressourcenbedarf und möglicherweise auch die erreichte Qualität.

Das Verfahren ist daher besser als Familie darstellbar:

```
V = f(Aufgabe, Qualitätsziel, Systemzustand, Sicherheitsvertrag)
```

Die Funktion `f` ist der Untersuchungsgegenstand.

Sechs Ebenen algorithmischer Verbesserung

„Algorithmische Verbesserung“ darf nicht mit einem schnelleren Kernel gleichgesetzt werden. Sie kann auf sechs Ebenen stattfinden.

1. **Implementierung:** Derselbe Operator wird besser vektorisiert, fusioniert oder speichereffizient ausgeführt.
2. **Ausführungsplan:** Bekannte Operatoren werden anders platziert, gepipelined oder zeitlich geordnet.
3. **Kommunikation:** Häufigkeit, Inhalt und Form der Synchronisation ändern sich.
4. **Topologie:** Knoten, Rollen, Repliken und Datenpartitionen werden neu gewählt.
5. **Lernverfahren:** Update-Regeln, lokale Arbeit und Umgang mit Asynchronität reagieren auf den Zustand.
6. **Modellform:** Sparsity, Modularität, externe Speicher oder bedingte Aktivierung verändern, was überhaupt berechnet wird.

Die ersten beiden Ebenen gehören zum etablierten Compiler- und Auto-Tuning-Feld. Die mittleren Ebenen berühren Scheduling und verteiltes Lernen. Die letzte Ebene greift in den Modellraum ein.

Project Ant Colony beginnt bewusst nicht auf Ebene sechs. Es öffnet die Freiheitsgrade schrittweise. Sonst wäre ein beobachteter Vorteil nicht zuordenbar.

Algorithmischer Freiheitsgrad

Der erste Zyklus hält Topologie und Zeitsteuerung fest. Variiert werden wenige, vorregistrierte Lern- und Kommunikationsentscheidungen.

Ein Beispiel ist die Zahl lokaler Schritte zwischen Synchronisationen. Bei `K = 1` kommunizieren Worker nach jedem Schritt. Größere Werte reduzieren die Synchronisationshäufigkeit, können aber dazu führen, dass lokale Modelle stärker auseinanderlaufen.

Local SGD ist keine Erfindung von Project Ant Colony. Die Forschung zeigt, dass lokale Updates Kommunikationsaufwand reduzieren können.[1] Die offene Frage ist enger:

> *Kann ein zustandsabhängiges System den geeigneten Wert unter wechselnder Bandbreite, Last und Worker-Geschwindigkeit besser wählen als eine starke feste oder regelbasierte Baseline?*

Ebenso werden Batchgröße, begrenzte Asynchronität, eine validierte Kompressionsoption und Straggler-Behandlung nicht frei erfunden, sondern aus einer geschlossenen Liste gewählt.

Topologischer Freiheitsgrad

Mehr Rechner bedeuten nicht automatisch mehr Geschwindigkeit.

Ein langsamer Worker kann eine synchrone Runde aufhalten. Ein Knoten mit wenig freiem Speicher kann häufiger fehlschlagen. Zusätzliche Teilnehmer erhöhen Kommunikation. Ein homogenes Teilcluster kann deshalb schneller sein als alle verfügbaren Maschinen.

Der topologische Freiheitsgrad umfasst:

- Auswahl aktiver Worker,
- Wahl des Koordinators,
- Datenaufteilung nach Kapazität oder Schrittzeit,
- Ausschluss eines ungeeigneten Knotens,
- und Aktivierung eines vorregistrierten Ersatzes.

Gelernte Clusterplanung ist bestehender Forschungsstand. Decima lernte workloadspezifische Scheduling-Policies für Datenverarbeitungscluster; Gandiva nutzte Laufzeitbeobachtung für Deep-Learning-Jobs.[2]

Project Ant Colony darf eine reine Platzierungsverbesserung daher nicht als neuen Lernalgorithmus ausgeben. Der mögliche Beitrag liegt in der Verbindung von Topologie mit Lern- und Kommunikationsentscheidungen unter demselben Live-Zustand.

Zeitlicher Freiheitsgrad

Ein Plan kann richtig sein und zu spät kommen.

Systemzustände altern. Hintergrundlast beginnt, ein Knoten wird heiß, ein Netzpfad wird langsamer. Zu häufige Neuplanung kostet jedoch selbst Zeit und kann Oszillation erzeugen.

Der zeitliche Freiheitsgrad bestimmt:

- wie lange Telemetrie aggregiert wird,
- wann eine Entscheidung ausgelöst wird,
- welche Abkühlzeit vor der nächsten Änderung gilt,

- wann ein Fehler als Ausfall zählt,
- und wann das System auf eine eingefrorene Baseline zurückfällt.

Damit wird Zeit nicht nur gemessen. Sie wird Teil der Strategie.

Auswahl vor Synthese

Die Vision eines KI-Systems, das neue Algorithmen erfindet, ist faszinierend und für den ersten Test ungeeignet.

AlphaTensor, AlphaDev, AutoML-Zero und AlphaEvolve zeigen, dass maschinell geführte Suche neue oder verbesserte Algorithmen finden kann, wenn Suchraum und Bewertung präzise genug sind.[3] Diese Arbeiten unterstreichen aber auch die Bedeutung eines maschinell prüfbaren Evaluators.

Project Ant Colony trennt daher drei Reifegrade:

1. **Auswahl:** Das Modell wählt nur aus versionierten gültigen Strategien.
2. **Parametrisierung:** Es kombiniert erlaubte Werte innerhalb definierter Grenzen.
3. **Synthese:** Erst nach bestandenen Mess- und Sicherheitsgates darf neuer Code vorgeschlagen werden.

In der hier eröffneten Forschung ist Synthese eine spätere Stufe, keine Fähigkeitsbehauptung.

Optimierung braucht eine Zielfunktion

„Schneller“ ist für ein Lernsystem unvollständig.

Ein Plan kann die Laufzeit senken, indem er weniger Qualität erreicht. Er kann Netzwerkverkehr sparen und dabei mehr Energie verbrauchen. Er kann einen Benchmark gewinnen, aber so viel menschliche Nacharbeit verlangen, dass der Vorteil verschwindet.

Die primäre Zielgröße lautet deshalb:

> *Netto-Zeit bis zu einer vorregistrierten Qualitätsgrenze.*

Netto bedeutet einschließlich Messung, Modellentscheidung, Validierung und Neuplanung.

Sekundär werden Kommunikation, Energie, Fehlerrate, menschlicher Aufwand und Robustheit berichtet. Eine Qualitäts- oder Sicherheitsverletzung kann nicht durch eine bessere Laufzeit kompensiert werden.

Der Körper antwortet

Die Umkehrung dieses Kapitels ist keine reine Softwareidee.

Das Verfahren wird vorgeschlagen, ausgeführt und gemessen. Der reale Körper antwortet. Vielleicht profitiert er von mehr lokalen Schritten. Vielleicht zerstört die

zusätzliche Drift den Vorteil. Vielleicht ist ein kleineres Teilcluster schneller. Vielleicht dominiert eine einfache Heuristik jedes gelernte Modell.

Erst die Rückmeldung trennt übersehene Möglichkeit von schöner Erzählung.

Damit die Antwort wissenschaftlich brauchbar ist, darf das trainierte Modell jedoch nicht direkt handeln.

Zwischen Vorschlag und Maschine braucht es einen Compiler für Entscheidungen.

Quellenhinweise

1 Sebastian U. Stich, „Local SGD Converges Fast and Communicates Little“, *International Conference on Learning Representations* (2019).

2 Hongzi Mao et al., „Learning Scheduling Algorithms for Data Processing Clusters“, *ACM SIGCOMM* (2019), 270-288; Wencong Xiao et al., „Gandiva: Introspective Cluster Scheduling for Deep Learning“, *13th USENIX Symposium on Operating Systems Design and Implementation* (2018), 595-610.

3 Alhussein Fawzi et al., „Discovering Faster Matrix Multiplication Algorithms with Reinforcement Learning“, *Nature* 610 (2022), 47-53; Daniel J. Mankowitz et al., „Faster Sorting Algorithms Discovered Using Deep Reinforcement Learning“, *Nature* 618 (2023), 257-263.

KAPITEL 16

Der Decision Compiler

Kein Gott-Kernel

Ein probabilistisches Modell schlägt vor; ein deterministischer Kern validiert, begrenzt und führt aus.

Ein trainiertes Modell kann plausible Systementscheidungen erzeugen. Es kann aus Telemetrie ableiten, welche Knoten für eine Aufgabe geeignet erscheinen, wie oft synchronisiert werden sollte oder wann eine Neuplanung sinnvoll wäre.

Es kann ebenso halluzinieren, veraltete Zustände verwenden, Regeln missverstehen und eine überzeugende Begründung für einen ungültigen Plan liefern.

Darum darf es kein autonomer Gott-Kernel sein.

Der Decision Compiler teilt das System in zwei erkenntnistheoretisch verschiedene Hälften:

- Ein trainiertes Modell arbeitet probabilistisch. Es schätzt, ordnet und schlägt vor.
- Ein deterministischer Kern arbeitet vertraglich. Er prüft, begrenzt, übersetzt und verwirft.

Die Wahrscheinlichkeit bleibt im Vorschlag. Ausführungsrecht entsteht erst an der harten Grenze.

Warum „Compiler“?

Ein gewöhnlicher Compiler übersetzt ein Programm aus einer Darstellung in eine andere und bewahrt dabei definierte Bedeutungen und Regeln.

Der Decision Compiler übersetzt keine Programmiersprache im engeren Sinn. Er übersetzt eine *Entscheidungsabsicht* in einen ausführbaren, begrenzten Plan.

Der Name betont drei Eigenschaften:

1. Die Modellausgabe ist noch keine Aktion.
2. Die Übersetzung folgt einem expliziten Vertrag.
3. Ungültige Eingaben werden abgelehnt, nicht kreativ repariert.

Ein Sprachmodell kann sagen: „Nutze die zwei schnellsten Worker und synchronisiere seltener.“ Der Decision Compiler benötigt stattdessen strukturierte Felder: konkrete Worker-IDs aus einer Positivliste, einen zulässigen Wert für lokale Schritte, einen Zeitstempel, eine Policy-Version und einen Fallback.

Die sechs Stationen

```
1. messen
      |
2. Zustand normalisieren und signieren
      |
3. Plankandidaten erzeugen oder bewerten
      |
4. deterministisch validieren
      |
5. begrenzt ausführen
      |
6. Ergebnis und Entscheidungsspur speichern
```

Jede Station besitzt eine andere Fehlerklasse.

Die Messung kann unvollständig sein. Der Zustandsvektor kann veraltet sein. Das Modell kann falsch priorisieren. Der Validator kann einen Vertrag unvollständig abbilden. Der Executor kann technisch scheitern. Die Auswertung kann einen Vorteil falsch zurechnen.

Eine monolithische „KI-Steuerung“ würde diese Fehler vermischen. Die Schichtung macht sie beobachtbar.

Der spezialisierte MOS-Kopf

Das Model OS, kurz MOS, muss kein allgemeiner Chatbot und kein universeller Coder sein.

Als eigenes trainiertes Modell hätte es eine engere Domäne:

- Eignung von Knoten,
- Auswahl verteilter Strategien,
- Vorhersage von Laufzeit und Kommunikationskosten,
- Erkennung von Zustandswechseln,
- und Priorisierung gültiger Plankandidaten.

Seine Eingaben wären strukturierte Merkmale, nicht primär freie Sprache:

```
Aufgabe:
  Modellklasse, Datenmenge, Qualitätsziel, Zeitbudget

Knoten:
  Capability, freie Kerne, freier RAM, Schrittzeit, Last

Netz:
  Durchsatz, RTT, Verlust, letzte Messung

Verlauf:
  ähnliche Pläne, Ergebnis, Fehler, Overhead
```

Die Ausgabe wäre ein Ranking oder deklarativer Plan.

Ein kleines Modell kann für diese Aufgabe besser geeignet sein als ein großer Generalist. Es ist günstiger messbar, leichter einzufrieren und kann auf lokalen Erfahrungsdaten trainiert werden. Ob es tatsächlich besser ist, bleibt eine Experimentfrage.

Der Planvertrag

Ein gültiger Plan enthält mindestens:

- Schema- und Policy-Version,
- Hash des verwendeten Snapshots,
- Ausgabe- und Ablaufzeit,
- aktive Worker und optionalen Koordinator,
- lokale Schritte und globalen Batch,
- Synchronisations- und Kompressionsmodus,
- Entscheidungsintervall und Replan-Cooldown,
- sowie einen eingefrorenen Fallback-Plan.

Die Felder besitzen geschlossene Wertebereiche. Unbekannte Worker, überalterte Snapshots, nicht geprüfte Kompressionsverfahren oder fehlende Fallbacks führen zur Ablehnung.

Freitext darf die Entscheidung erklären. Er besitzt keine Ausführungswirkung.

Diese Einschränkung ist keine Schwäche des Modells. Sie ist die Versuchsbedingung, unter der sein Beitrag messbar wird.

Was der Validator nicht darf

Auch die deterministische Hälfte benötigt Grenzen.

Ein Validator darf einen ungültigen Vorschlag nicht heimlich in einen guten Plan umschreiben. Sonst wäre unklar, ob das Modell oder die Reparaturlogik den Erfolg erzeugte.

Er entscheidet nur:

- gültig,
- ungültig mit Grund,
- abgelaufen,
- oder wegen fehlender Kapazität nicht ausführbar.

Ein verworfener Plan ist ein Ergebnis. Seine Häufigkeit gehört zur Modellbewertung.

Ausführung unter Vorbehalt

Nach der Validierung setzt ein begrenzter Executor den Plan um.

Er darf nur vorregistrierte Aktionen ausführen. CPU-, RAM-, Laufzeit- und Netzwerkgrenzen werden außerhalb des Modells erzwungen. Produktive Dienste

besitzen Vorrang. Bei Telemetrieverlust, Grenzwertverletzung oder kontrolliertem Fehler übernimmt der Fallback.

Die Ausführungsgrenze ist absichtlich konservativ. Project Ant Colony untersucht nicht, ob ein Agent möglichst autonom sein kann. Es untersucht, ob seine Entscheidung innerhalb harter Grenzen zusätzlichen Nutzen besitzt.

Forschungsnachbarn

Der Decision Compiler steht nicht außerhalb vorhandener Forschung.

LLVM MLGO ersetzt oder ergänzt ausgewählte Compilerheuristiken durch gelernte Modelle. Die aktuelle LLVM-Dokumentation nennt unter anderem Inlining für Codegröße und Registerallokation; korrektheitserhaltende Regeln bleiben von Optimierungsentscheidungen getrennt.[1]

TVM und Ansor optimieren Tensorprogramme mit Suchverfahren und gelernten Kostenmodellen. OpenTuner organisiert domänenspezifische Auto-Tuning-Suchräume. Decima lernt Scheduling-Policies. AlphaTensor, AlphaDev und AlphaEvolve verbinden maschinelle Suche mit automatisierten Evaluatoren.[2]

Aus diesen Nachbarn folgt keine Neuheit von MOS.

Die Kandidatenlücke liegt in ihrer Verbindung:

> *live gemessener Zustand eines heterogenen Commodity-Clusters, mehrere Entscheidungsebenen, ein begrenztes trainiertes Modell, deklarative Pläne und eine deterministische Ausführungsgrenze.*

Bis eine systematische Recherche diese Kombination geprüft hat, bleibt sie eine Kandidatenlücke.

Lernen aus lokalen Spuren

Ein MOS-Modell benötigt Daten.

Jeder gültige Lauf erzeugt ein Beispiel:

- Zustand,
- gewählter Plan,
- vorhergesagter Effekt,
- tatsächliche Zeit bis Qualität,
- Kommunikation,
- Fehler,
- und Entscheidungsoverhead.

Ungültige Vorschläge erzeugen ebenfalls Beispiele. Sie lehren, welche Planregionen am Vertrag scheitern.

Das Training kann zunächst als Kostenmodell oder Rankingproblem formuliert werden. Für einen gegebenen Zustand ordnet das Modell gültige Kandidaten. Diese Form ist kontrollierbarer als freie Codesynthese.

Später kann die Suche neue Parametervarianten oder Module vorschlagen. Der Evaluator und die Sicherheitsgrenze bleiben gleich.

Was Erfolg bedeuten würde

Ein guter Decision Compiler muss nicht spektakulär wirken.

Er könnte zeigen, dass bei hoher Heterogenität ein Teilcluster schneller ist. Er könnte ein Synchronisationsintervall wählen, das Kommunikation spart, ohne die Qualitätsgrenze zu verletzen. Er könnte rechtzeitig auf einen sicheren Plan zurückfallen.

Sein Erfolg wird nicht an der Eleganz der Begründung gemessen, sondern an vier Fragen:

1. War der Plan gültig?
2. War er auf den verwendeten Zustand rückführbar?
3. Übertraf er die stärkste zulässige Baseline nach Abzug des Overheads?
4. Wiederholte sich der Vorteil auf gehaltenen Szenarien?

Wenn eine feste Regel gleich gut ist, braucht es das Modell nicht.

Gerade diese Möglichkeit macht den Decision Compiler zu Forschung statt Produktversprechen.

Quellenhinweise

1 LLVM Project, „Machine Learning - Guided Optimization (MLGO)“, offizielle LLVM-Dokumentation, Stand 2026.

2 AlphaEvolve Team, „AlphaEvolve: A Gemini-Powered Coding Agent for Designing Advanced Algorithms“, Google DeepMind, 14. Mai 2025.

KAPITEL 17

Das Ökosystem wird zur Variablen

Eine Maschine ist kein Typenschild

Capability, Last, Netzwerk und Zeit verwandeln den Hardwaretyp in einen lebenden, gültigen Zustand.

Ein Inventar kann sagen, welcher Prozessor in einem Rechner steckt, wie viel RAM installiert ist und welches Netzwerkgerät erkannt wurde.

Für eine Ausführungsentscheidung genügt das nicht.

Ein Knoten mit acht logischen CPUs kann durch Gastmaschinen belastet sein. Freier Speicher kann innerhalb von Minuten verschwinden. Ein nomineller Gigabit-Link kann durch konkurrierenden Verkehr langsam werden. Eine integrierte GPU kann sichtbar sein, ohne dass ein stabiler Rechenpfad existiert.

Die Eignung eines Geräts ist keine dauerhafte Eigenschaft.

> *Nicht die gespeicherte Hardwareklasse entscheidet, sondern der gültige gemessene Zustand.*

Der Snapshot

Project Ant Colony arbeitet deshalb mit zeitgestempelten Snapshots.

Ein Snapshot verbindet statische und dynamische Merkmale:

- Knotenklasse und bestätigte Capabilities,
- physische Kerne und freier Speicher,
- aktuelle CPU- und Gastlast,
- gemessene Worker-Schrittzeit,
- Netzwerkdurchsatz und Latenz,
- gesendete Bytes und Alter der letzten Synchronisation,
- Validierungsmetrik,

- Zeitpunkt und Gültigkeitsfenster.

Der Snapshot erhält eine Identität, idealerweise einen kryptografischen Hash. Jeder Plan verweist auf genau diesen Zustand.

Ist ein Pflichtwert unbekannt oder der Snapshot zu alt, darf das Modell nicht so tun, als sei der günstige Wert wahrscheinlich. Der Zustand ist ungültig.

Fähigkeit vor Gerätebezeichnung

Der Name eines Geräts ist ein Hinweis, kein Vertrag.

Zwei Rechner desselben Modells können unterschiedliche Speicherbestückung, thermische Bedingungen, Softwarestände und Hintergrundlast besitzen. Eine PCI-Kennung belegt, dass ein Grafikgerät existiert. Sie belegt weder Korrektheit noch Stabilität eines ML-Backends.

Project Ant Colony verwendet daher Capability-Gates:

1. Backend vorhanden?
2. Referenzoperation korrekt?
3. Ergebnis über Wiederholungen stabil?
4. Ressourcenlimits durchsetzbar?
5. Messkanäle vollständig?

Erst danach wird eine Ressource in den Aktionsraum aufgenommen.

Diese Reihenfolge schützt vor einem typischen Systemfehler: theoretische Kapazität wird als praktisch verfügbare Leistung gezählt.

Der lebende Verbund

Der sanitierte Live-Snapshot vom 14. Juli 2026 erfasste acht Proxmox-Mitglieder, davon sieben online. Fünf Online-Knoten wurden als `Macmini6,2`, zwei als `Macmini7,1` erkannt. Zusammen besaßen sie 24 physische Kerne, 48 logische CPUs und rund 100,8 GiB RAM.[1]

Diese Summe ist keine verfügbare Forschungskapazität.

Mehrere Knoten trugen laufende Gäste. Ein Mitglied war offline. Die integrierten Intel-Grafikeinheiten waren als Geräte belegt, ihre Trainingsfähigkeit jedoch nicht. Ein separater MLX-Host war als möglicher Planungskopf dokumentiert, seine genaue Hardware noch nicht live bestätigt.

Das Inventar zeigt damit nicht nur Ressourcen. Es zeigt Unsicherheit, Ausschlüsse und Schutzbedarf.

Die Plattform ist ein Ökosystem, kein leerer Cluster.

Drei Arten von Kapazität

Für jedes Gerät müssen drei Kapazitäten getrennt werden.

Physische Kapazität ist installiert: Kerne, Speicher, Laufwerke, Netzwerk.

Zugewiesene Kapazität ist bereits durch Hypervisor, Gäste oder andere Dienste gebunden.

Freigegebene Forschungskapazität ist der Teil, den ein Experiment in einem definierten Zeitfenster tatsächlich nutzen darf.

Nur die dritte Größe gehört in den Executorvertrag.

Ein Modell, das physische Summen optimiert, plant auf einer Maschine, die nicht existiert.

Topologie als Momentaufnahme

In klassischen Diagrammen ist eine Cluster-Topologie stabil. Knoten sind Kästchen, Verbindungen sind Linien.

Im Experiment ist Topologie ein zeitabhängiger Graph. Ein Knoten kann online und dennoch unzulässig sein. Eine Verbindung kann existieren und für den aktuellen Plan zu langsam sein. Ein Worker kann teilnehmen, aber seine Schrittzeit die Gruppe dominieren.

Der Decision Compiler muss deshalb zwischen vier Zuständen unterscheiden:

- erreichbar,
- fähig,
- freigegeben,
- und für den konkreten Plan geeignet.

Nur ein Knoten, der alle vier Bedingungen erfüllt, ist ein Kandidat.

Das Netzwerk spricht mit

Auf einem Gigabit-Verbund kann Kommunikation den Nutzen zusätzlicher Worker vollständig aufzehren.

Darum werden nicht nur Linknamen erfasst. Benötigt werden wiederholte Messungen von Durchsatz, Round-Trip-Zeit, Verlust und Schwankung - im ruhigen Zustand und im geplanten Forschungsfenster.

Ein einzelner Bestwert ist wertlos. Das Modell benötigt Verteilungen oder robuste Zusammenfassungen, weil der langsamste Pfad eine synchrone Runde bestimmen kann.

Die Kommunikation wird so von einer Implementierungskennzahl zur Entscheidungsvariable:

- mehr lokale Schritte,
- weniger Worker,
- andere Datenpartition,
- Kompression,
- oder zeitliche Verschiebung.

Zeitstempel als Sicherheitsmerkmal

Ein guter Plan für gestern kann heute falsch sein.

Jeder Snapshot erhält deshalb ein maximales Alter. Jeder Plan besitzt Ausgabe- und Ablaufzeit. Zwischen Vorschlag und Ausführung prüft der Validator, ob der referenzierte Zustand noch gültig ist.

Diese Regel klingt banal. Sie trennt jedoch zwei Arten von Intelligenz:

- ein Modell, das auf abstrakte Hardwareeigenschaften reagiert;
- ein System, das unter realer Veränderung verantwortlich handeln kann.

Project Ant Colony untersucht die zweite.

Das Ökosystem lernt nicht von selbst

Messung allein erzeugt keine gute Entscheidung.

Ein immer größerer Zustandsvektor kann das Modell überfordern. Viele Merkmale sind korreliert. Manche Messungen kommen zu spät. Andere verändern das System, das sie beobachten. Ein Telemetriedienst besitzt eigenen CPU-, Speicher- und Netzwerkbedarf.

Darum muss jedes Merkmal eine Begründung besitzen:

- Welche Entscheidung kann sich dadurch ändern?
- Wie wird es gemessen?
- Wie alt darf es sein?
- Was geschieht bei Ausfall?
- Gehört sein Overhead zur Bilanz?

Ein Zustand ist kein Datenfriedhof. Er ist ein Vertrag zwischen Beobachtung und Handlung.

Von der Maschine zur Umwelt

Sobald Last, Verfügbarkeit und Zeit Teil der Entscheidung werden, verändert sich die Bedeutung von Hardware.

Sie ist nicht mehr das Ziel, für das einmal optimiert wird. Sie ist eine Umwelt, auf die ein Verfahren fortlaufend reagiert.

Diese Sicht führt zur Ameisenmetapher. Eine Kolonie besitzt keinen allwissenden Arbeiter, der jede Veränderung vorausberechnet. Viele begrenzte Einheiten arbeiten in einer widerspenstigen Umgebung; Spuren und lokale Rückmeldungen verändern das gemeinsame Verhalten.

Project Ant Colony übernimmt davon nicht blind einen biologischen Algorithmus. Es übernimmt die Forschungsintuition:

> *Intelligenz kann in der Passung zwischen begrenzten Akteuren, messbaren Spuren und einer veränderlichen Umwelt liegen.*

Quellenhinweise

1 Project Ant Colony, *Hardware-Inventur und Versuchstauglichkeit*, sanitierter Live-Snapshot vom 14. Juli 2026, Primärartefakt `hardware_inventory_live_2026-07-14.json`.

KAPITEL 18

Das Ameisenmodell als Forschungsinstrument

Die Rechner sind nicht die Behauptung

Die heterogene Althardware ist kein Ersatzversprechen, sondern ein absichtlich schwieriger Windkanal.

Project Ant Colony besteht aus älteren Commodity-Rechnern. Das lädt zu einer falschen Erzählung ein:

Viele kleine CPUs sollen zeigen, dass spezialisierte Beschleuniger überflüssig sind.

Das ist nicht die These.

Für dichte Tensoroperationen besitzen moderne Beschleuniger klare physische Vorteile. Ein Verbund alter Rechner kann Speicherbandbreite, spezialisierte Recheneinheiten und schnelle Geräteverbindungen nicht durch Metaphern ersetzen.

Die alten Maschinen sind nicht die Behauptung. Sie sind der Windkanal.

Warum ein absichtlich schwieriger Prüfstand?

Ein guter Windkanal erzeugt kontrollierte Widerstände.

Der Ant-Colony-Verbund besitzt genau jene Eigenschaften, die im reifen Beschleunigerstack gern unsichtbar werden:

- unterschiedliche CPU-Generationen und Kernzahlen,
- knapper und ungleich verteilter Speicher,
- Gigabit-Ethernet statt schneller Accelerator-Fabrics,
- laufende Hintergrundlast,
- zeitweise nicht verfügbare Knoten,
- unbewiesene Gerätepfade,
- und strenge Schutzgrenzen für produktive Dienste.

Unter diesen Bedingungen kann ein Verfahren seine Annahmen nicht verstecken.

Wenn jeder Schritt globale Synchronität verlangt, wird die Netzgrenze sichtbar. Wenn der langsamste Worker alle aufhält, wird Topologie sichtbar. Wenn ein Plan veraltet, wird Zeit sichtbar. Wenn Telemetrie fehlt, wird Sicherheit sichtbar.

Der Prüfstand vergrößert die Phänomene, die untersucht werden sollen.

Die konkrete Kolonie

Der Live-Snapshot vom 14. Juli 2026 dokumentierte sieben erreichbare Proxmox-Knoten aus zwei Mac-mini-Generationen. Fünf Geräte gehörten zur Klasse `Macmini6,2`, zwei zu `Macmini7,1`. Gemeinsam wurden 24 physische Kerne, 48 logische CPUs und rund 100,8 GiB RAM erfasst.[1]

Die Knoten waren nicht homogen. Zwei Geräte besaßen nur zwei physische Kerne, eines davon rund 7,64 GiB RAM. Mehrere Hosts trugen laufende Gäste. Ein achtes Clustermitglied war offline.

Diese Unterschiede sind keine störende Abweichung von einem idealen Benchmark. Sie sind der Gegenstand.

Gleichzeitig werden sie kontrolliert. Ein Experiment darf nur vorregistrierte, freigegebene Knoten nutzen. Hintergrundlast wird dokumentiert. Spontane Änderungen, die einen Lauf retten sollen, sind unzulässig.

Warum Ameisen?

Eine Ameisenkolonie verbindet drei Eigenschaften, die als Denkfigur nützlich sind.

Erstens sind die einzelnen Einheiten begrenzt. Keine Arbeiterin trägt das vollständige Modell der Kolonie.

Zweitens entsteht Koordination teilweise indirekt über Spuren in der Umwelt. In der Informatik wurde diese Idee in Ant-Colony-Optimization formalisiert: künstliche Agenten bauen Lösungen und beeinflussen spätere Suche über akkumulierte Pheromonspuren.[2]

Drittens ist Robustheit verteilt. Der Ausfall einer Einheit beendet nicht notwendig das Gesamtsystem.

Project Ant Colony ist dennoch nicht einfach ein Ant-Colony-Optimization-Algorithmus. Es setzt nicht voraus, dass Jobs wie Pheromonpfade behandelt werden oder dass biologische Selbstorganisation die beste Schedulerlogik liefert.

Die Ameise ist hier zuerst eine epistemische Figur:

> *Begrenzte lokale Akteure können gemeinsam einen größeren Suchraum erschließen, wenn ihre Spuren messbar und ihre Regeln klar sind.*

Die Spur ist das Artefakt

Im Forschungsinstrument wird die Pheromonmetapher durch Protokolle ersetzt.

Jeder Lauf hinterlässt:

- den verwendeten Snapshot,
- die Policy- und Schemaversion,
- den vorgeschlagenen Plan,
- das Validatorurteil,
- Executor-Ereignisse,
- Rohmetriken,
- Fehler,
- und einen unveränderlichen Ergebnisstatus.

Diese Spur hat zwei Funktionen.

Wissenschaftlich macht sie Entscheidungen reproduzierbar. Ein späterer Leser kann prüfen, welcher Zustand zu welchem Plan führte.

Für das Modell bildet sie Erfahrung. Ähnliche Zustände und ihre gemessenen Folgen können zur Priorisierung neuer Kandidaten verwendet werden.

Die Spur ist damit Gedächtnis, aber kein Mythos. Sie besteht aus prüfbaren Artefakten.

Zentraler Vorschlag, dezentrale Arbeit

Die Kolonie ist nicht vollständig dezentral.

Ein MOS-Kopf kann globalen Zustand zusammenführen und Pläne vorschlagen. Ein Validator und Executor setzen zentrale Verträge durch. Die Worker führen lokale Arbeit aus.

Diese Hybridform ist absichtlich pragmatisch. Vollständige Dezentralität wäre eine zusätzliche Forschungsfrage. Sie würde Konsens, verteilte Fehlertoleranz und Sicherheitskoordination in den Pilot hineinziehen.

Project Ant Colony untersucht zunächst:

> *Kann ein begrenzter Entscheidungskopf die verteilte lokale Arbeit besser an den Zustand anpassen als feste oder regelbasierte Strategien?*

Die Ameisenmetapher bestimmt also nicht die Architektur. Sie schützt nur vor der Vorstellung, Intelligenz müsse in einem einzigen großen Rechenkörper liegen.

Wiederverwendung ohne grüne Romantik

Vorhandene Hardware weiterzuverwenden kann Beschaffung vermeiden. Daraus folgt keine automatische ökologische Überlegenheit.

Alte Rechner können für dieselbe Aufgabe länger laufen und mehr Energie verbrauchen. Ein Verbund benötigt Netzwerk, Speicher und Betriebsaufwand. Ein externer Planungskopf kommt hinzu.

Darum darf Project Ant Colony Nachhaltigkeit nur behaupten, wenn Energie bis zum Qualitätsziel belastbar gemessen und der Lebenszyklusrahmen offengelegt wird.

Fehlt ein kalibrierter Zähler, lautet der Energiestatus `missing-not-measured`. CPU-Auslastung wird nicht in eine scheinpräzise Energiezahl umgerechnet.

Die Wiederverwendung ist zunächst eine Zugangs- und Forschungsbedingung, kein Umweltsiegel.

Abgrenzung zu Der Körper der KI

Der Körper der KI behandelt Infrastruktur, Betrieb und materielle Bedingungen künstlicher Intelligenz auf einer breiteren Ebene.

Project Ant Colony schließt thematisch daran an, bleibt aber ein eigenständiges Buch.

Hier ist die Kolonie kein allgemeines Betriebsmodell für KI. Sie ist ein konkretes Forschungsinstrument für eine engere Frage: Kann ein KI-gestützter, auditierbarer Entscheidungsprozess algorithmische, topologische und zeitliche Freiheitsgrade auf heterogener Althardware produktiv nutzen?

Das Buch setzt die Kenntnis des anderen Werks nicht voraus. Die Verbindung liegt in einem gemeinsamen Grundgedanken:

Auch digitale Intelligenz besitzt einen Körper, und dieser Körper wählt mit.

Was der Prüfstand leisten kann

Der Cluster kann keine Aussagen über das Training großer Foundation Models tragen.

Er kann nicht beweisen, dass CPUs GPUs ersetzen.

Er kann auch nicht ohne Weiteres auf moderne Edge-Cluster, Rechenzentren oder andere Netzwerke generalisiert werden.

Er kann etwas Bescheideneres und wissenschaftlich Nützliches leisten:

- Kippunkte zwischen Kommunikation und lokaler Arbeit sichtbar machen,
- die Wirkung heterogener Worker kontrolliert testen,
- den Overhead eines Decision Compilers messen,
- ungültige Pläne und Rückfallverhalten dokumentieren,
- und starke Baselines unter identischen Bedingungen vergleichen.

Wenn eine gelernte Policy gewinnt, zeigt der Prüfstand einen begrenzten zustandsabhängigen Vorteil.

Wenn sie verliert, zeigt er, wo die alternative Evolutionsidee an realen Kosten scheitert.

Vom Instrument zur Forschungsfrage

Bis hierhin hat das Buch einen langen Weg zurückgelegt.

Eine historische Abzweigung führte zu mehreren möglichen Welten. Diese Welten führten zu einer veränderten Modellintuition. Daraus entstand die Umkehrung von Hardware- und Algorithmusanpassung. Der Decision Compiler machte die Umkehrung sicher ausführbar. Das Ameisenmodell gibt ihr einen widerständigen Körper.

Nun muss die Erzählung eng werden.

Nicht mehr: Könnte eine andere KI-Evolution möglich gewesen sein?

Sondern:

> *Welche einzelne, falsifizierbare Frage kann dieser konkrete Prüfstand beantworten?*

Quellenhinweise

1 Project Ant Colony, *Hardware-Inventur und Versuchstauglichkeit*, sanitierter Live-Snapshot vom 14. Juli 2026.

2 Marco Dorigo und Luca Maria Gambardella, „Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem“, *IEEE Transactions on Evolutionary Computation* 1, Nr. 1 (1997), 53-66, DOI: 10.1109/4235.585892.

TEIL V

Von der Erzählung zur Forschung

Forschungsfrage, Baselines, Messung und negative Ergebnisse machen den Evolutionsstrang falsifizierbar.

Kapitel 19 bis 22

KAPITEL 19

Die enge Forschungsfrage

Was von der großen Geschichte übrig bleibt

Aus der alternativen Geschichte wird eine kleine, verständliche und widerlegbare Forschungsfrage.

Das Buch begann mit einer Frage, die zu groß war:

Was wäre aus künstlicher Intelligenz geworden, wenn CUDA nicht zum erfolgreichsten Beschleuniger-Highway geworden wäre?

Diese Frage kann keine Forschung abschließend beantworten. Es gibt keinen zweiten Verlauf der Geschichte, mit dem sich unsere Welt vergleichen ließe. Kein Experiment kann beweisen, welche Unternehmen, Betriebssysteme oder Modelle in dieser anderen Welt entstanden wären.

Trotzdem war das Gegenfaktum nicht nutzlos.

Es hat sichtbar gemacht, was in der Gegenwart leicht wie eine Selbstverständlichkeit wirkt. Modelle werden meist für eine bekannte Ausführungsumgebung gebaut. Schnelle Verbindungen gelten als verfügbar. Bibliotheken verbergen viele Unterschiede. Die passende Hardware wird gesucht oder gemietet.

Die alternative Geschichte hat diese Richtung umgekehrt.

Was geschieht, wenn die vorhandene Umwelt nicht ausgetauscht werden kann? Wenn Geräte ungleich, Verbindungen langsam und Zustände wechselhaft sind? Dann wird nicht nur die Hardware zum Ziel eines festen Verfahrens. Auch das Verfahren muss sich bewegen können.

Aus dieser Beobachtung entsteht die Forschungsfrage.

Die Frage in einfacher Form

Project Ant Colony fragt:

> *Kann ein begrenztes, lernendes Entscheidungssystem aus dem aktuellen Zustand mehrerer ungleicher Rechner einen guten Ausführungsplan wählen, ohne die Kontrolle über Sicherheit, Qualität und Nachvollziehbarkeit zu erhalten?*

Die Formulierung ist absichtlich klein.

Das System soll nicht beweisen, dass alte CPUs moderne Beschleuniger ersetzen. Es soll kein allgemeines Betriebssystem erfinden. Es soll auch nicht selbstständig beliebige Programme auf fremden Rechnern ausführen.

Es soll eine überschaubare Entscheidung treffen.

Vielleicht nutzt eine Aufgabe nur einen schnellen Knoten. Vielleicht arbeiten mehrere langsamere Knoten gemeinsam. Vielleicht lohnt sich zusätzliche lokale Arbeit, bevor Ergebnisse ausgetauscht werden. Vielleicht ist es besser, auf einen überlasteten Rechner zu verzichten.

Die richtige Wahl kann sich ändern, obwohl die Hardware dieselbe bleibt.

Genau darin liegt der mögliche Wert des gemessenen Zustands.

Ein Vorschlag ist noch keine Handlung

Ein trainiertes Modell kann Wahrscheinlichkeiten, Rangfolgen und plausible Pläne erzeugen. Das macht es nützlich. Es macht es aber nicht zuverlässig genug, um allein über die Ausführung zu entscheiden.

Project Ant Colony trennt deshalb Vorschlag und Handlung.

Das Modell darf sagen: Nutze diese Knoten, arbeite dort länger lokal und synchronisiere seltener.

Eine deterministische Schicht prüft anschließend, ob diese Wahl erlaubt ist. Sind die Knoten tatsächlich verfügbar? Bleibt genug Speicher? Wurde die Strategie freigegeben? Gibt es einen sicheren Rückfallplan?

Nur ein gültiger Plan darf ausgeführt werden.

Diese Trennung ist mehr als eine Sicherheitsmaßnahme. Sie macht das Ergebnis lesbar. Wenn ein Versuch gewinnt oder verliert, kann später rekonstruiert werden, welchen Zustand das Modell sah, welchen Plan es vorschlug und welche Grenze den Plan bestätigte oder verwarf.

Das trainierte System bleibt kreativ, aber es erhält keine unkontrollierte Autorität.

Warum gerade alte und ungleiche Rechner?

Der lokale Rechnerverbund ist kein Modell für das Internet und kein Ersatz für ein Rechenzentrum.

Er ist ein Windkanal.

Seine Geräte sind langsam genug, dass Kommunikation sichtbar wird. Sie sind ungleich genug, dass die Nutzung aller Knoten nicht automatisch die beste Wahl ist. Einige tragen bereits andere Dienste. Zustände verändern sich. Netzwerk, Speicher und Zeit lassen sich nicht als unsichtbare Selbstverständlichkeiten behandeln.

Auf perfekter Hardware könnte eine kluge Entscheidung kaum von einer einfachen Standardregel zu unterscheiden sein. Im absichtlich schwierigen Prüfstand werden die Unterschiede größer.

Das macht den Prüfstand interessant, aber nicht allgemein gültig.

Ein Erfolg würde zunächst nur zeigen, dass zustandsabhängige Auswahl unter diesen Bedingungen nützlich sein kann. Ein Misserfolg könnte zeigen, dass eine einfache Regel genügt, dass der Entscheidungsaufwand zu groß ist oder dass zusätzliche Rechner nur zusätzliche Last erzeugen.

Beides wäre eine Antwort.

Was an dieser Forschung nicht neu ist

Viele Bestandteile besitzen bereits eine Forschungsgeschichte.

Compiler können Programme an Hardware anpassen. Scheduler verteilen Arbeit. Auto-Tuning sucht schnelle Varianten. Lokale Lernverfahren reduzieren Kommunikation. Gelernte Kostenmodelle helfen bei Optimierungsentscheidungen. Systeme wie TVM, Ansor, MLGO, Decima und Gandiva stehen in dieser Nachbarschaft.

Project Ant Colony darf daraus keine breite Neuheitsbehauptung machen.

Die mögliche Lücke liegt in der Verbindung:

- ein zeitlich gültiger Zustand statt eines bloßen Gerätetyps,
- mehrere ungleiche Rechner statt eines homogenen Clusters,
- ein lernender Vorschlag statt einer festen Regel,
- eine deterministische Ausführungsgrenze,
- und ein Vergleich, in dem die einfachste gute Alternative gewinnen darf.

Auch diese Verbindung ist zunächst nur eine Kandidatenlücke. Eine gründliche Literaturprüfung oder ein externer Review kann zeigen, dass sie enger formuliert werden muss.

Wissenschaftliche Redlichkeit beginnt dort, wo eine Idee ihr eigenes Verschwinden als Neuheit zulässt.

Der kleinste sinnvolle Erfolg

Der Traum einer neuen Softwareevolution darf die Messlatte nicht verzerren.

Der kleinste sinnvolle Beitrag wäre keine universelle KI und kein selbstoptimierendes Weltbetriebssystem.

Er könnte viel bescheidener sein:

- eine nachvollziehbare Entscheidung, die eine starke feste Regel unter einer bestimmten Störung übertrifft,
- ein messbarer Punkt, an dem weniger Kommunikation wichtiger wird als mehr Rechenleistung,
- die Erkenntnis, dass ein Teil der Rechner besser arbeitet als der ganze Verbund,
- oder ein sauberer Negativbefund, der zeigt, dass das trainierte Modell keinen Zusatznutzen besitzt.

Solche Ergebnisse sind klein genug, um glaubwürdig zu sein, und allgemein genug, um neue Fragen zu erzeugen.

Wo das Buch endet

Bis hierher hat das Buch erklärt, warum diese Forschungsfrage existiert.

Es hat die technischen Brüche rekonstruiert, ihre gesellschaftlichen Bruchfolgen verfolgt, eine alternative Softwarewelt geöffnet und daraus eine begrenzte Architektur abgeleitet.

Nun beginnt eine andere Textform.

Die späteren Forschungspapiere müssen Aufgaben, Messwerte, Vergleichsverfahren und technische Implementierung vollständig offenlegen. Sie müssen die stärksten Baselines benennen, Fehler zählen und negative Ergebnisse bewahren.

Dieses Buch übernimmt nicht ihre Aufgabe.

Es muss nur noch erklären, woran eine faire Antwort zu erkennen wäre.

Damit führt die enge Forschungsfrage zum nächsten Schritt: Messung statt Vision.

KAPITEL 20

Messung statt Vision

Woran ein guter Ausgang zu erkennen wäre

Faire Vergleiche und die Zeit bis zu einem brauchbaren Ergebnis ersetzen die bloße Vision eines adaptiven Vorteils.

Eine Idee kann überzeugend klingen und trotzdem an der Wirklichkeit vorbeigehen.

Vielleicht wählt das Modell einen eleganten Plan, braucht für die Entscheidung aber länger als der Plan einspart. Vielleicht sinkt die Laufzeit, weil die Qualität schlechter wird. Vielleicht gewinnt der Verbund nur gegen eine absichtlich schwache Vergleichsmethode. Vielleicht funktioniert alles einmal und nie wieder.

Darum genügt die Vision eines anpassungsfähigen Systems nicht.

Die Forschung benötigt einen fairen Vergleich.

Für das Buch muss dieser Vergleich noch nicht bis in jede Formel beschrieben werden. Diese Arbeit gehört in die späteren Papiere. Aber seine Grundidee muss verständlich sein, denn ohne sie bliebe auch die historische Erzählung ohne Widerstand.

Nicht Geschwindigkeit, sondern ein brauchbares Ergebnis

Der naheliegende Messwert wäre Geschwindigkeit.

Doch eine schnellere Ausführung ist nicht automatisch besser. Eine Übersetzung, die entscheidende Sätze verliert, ist nicht dadurch gut, dass sie früher endet. Ein Modell, das weniger Daten verarbeitet und deshalb ungenauer wird, hat keinen Fortschritt erzeugt.

Verglichen werden muss deshalb die Zeit bis zu einem zuvor bestimmten, brauchbaren Ergebnis.

Die Aufgabe bleibt gleich. Die Daten bleiben gleich. Die gewünschte Qualität bleibt gleich. Erst dann darf gefragt werden, welcher Weg dieses Ziel schneller oder mit weniger Aufwand erreicht.

Dieser einfache Gedanke schützt das Projekt vor einer häufigen Versuchung: Man darf die Ziellinie nicht nachträglich näher heranrücken, nur weil der eigene Läufer müde wird.

Der stärkste ehrliche Gegner

Eine neue Methode kann leicht gegen einen schlechten Vergleich gewinnen.

Wenn der lernende Agent gegen eine naive Verteilung aller Aufgaben auf alle Rechner antritt, sagt ein Sieg wenig aus. Vielleicht hätte eine erfahrene Systemadministratorin dieselbe Fehlentscheidung sofort vermieden. Vielleicht reicht eine einfache Regel: Nutze langsame Knoten nur, wenn das Netzwerk frei ist.

Darum muss der stärkste bekannte einfache Gegner mitlaufen.

Das können feste Strategien, gut abgestimmte Regeln, vorhandene Auto-Tuning-Verfahren oder etablierte Scheduler sein. Sie erhalten denselben Zugriff auf die Aufgabe und denselben Tuningaufwand. Sie dürfen gewinnen.

Der Decision Compiler ist nur dann gerechtfertigt, wenn sein zusätzlicher Aufwand einen zusätzlichen Nutzen erzeugt.

Wenn eine Schwellenregel gleich gut ist, gehört der Sieg der Schwellenregel.

Der Preis der Entscheidung zählt mit

Auch ein kluger Plan besitzt Kosten.

Der Zustand muss gemessen werden. Das Modell benötigt Rechenzeit. Ein Vorschlag wird geprüft. Vielleicht muss neu geplant werden. Protokolle werden geschrieben. Menschen kontrollieren Ausnahmen.

All diese Arbeit gehört zum Ergebnis.

Ein System darf sich nicht nur mit der reinen Ausführungszeit schmücken und seinen eigenen Entscheidungsapparat verstecken. Der Vorteil muss übrig bleiben, nachdem Messung, Planung, Validierung und Rückfalllogik bezahlt sind.

Das ist besonders wichtig für kleine Aufgaben. Dort kann ein aufwendiger Entscheidungskopf mehr kosten als jede Optimierung, die er findet.

Auch diese Niederlage wäre nützlich. Sie würde zeigen, ab welcher Größe oder unter welcher Störung die Anpassung überhaupt sinnvoll wird.

Qualität und Sicherheit sind keine Handelsware

Ein schnellerer Lauf darf keine verbotene Abkürzung nehmen.

Er darf weder ein schlechteres Ergebnis liefern noch Speichergrenzen, Netzwerkregeln oder Sicherheitsvorgaben verletzen. Er darf keinen Rechner verwenden, der für den Versuch nicht freigegeben ist. Ein ungültiger Plan wird nicht kreativ repariert und anschließend als Erfolg gezählt.

Die Grenze muss hart bleiben.

Gerade ein lernendes System könnte sonst aus den Versuchen die falsche Lektion ziehen: Regelverletzung wird belohnt, solange sie Zeit spart.

Project Ant Colony trennt deshalb drei Dinge:

- Der Vorschlag kann neu und überraschend sein.
- Die Prüfung bleibt fest und nachvollziehbar.
- Die Ausführung findet nur innerhalb der erlaubten Grenze statt.

Diese Ordnung macht aus einer Demonstration ein Experiment.

Mehr als eine Zahl

Die spätere Forschung wird mehrere Wirkungen betrachten müssen.

Wie lange dauert es bis zum brauchbaren Ergebnis? Wie viele Daten wandern durch das Netzwerk? Wie viel menschliche Aufmerksamkeit wird benötigt? Wie oft wird neu geplant? Bleibt das System bei einer Störung ruhig oder wechselt es hektisch zwischen Strategien?

Auch Energie gehört dazu, aber nur, wenn sie belastbar gemessen werden kann. Eine grobe Schätzung aus Prozessorauslastung darf nicht zur Nachhaltigkeitsaussage werden.

Das Buch muss an dieser Stelle keine Metriktabelle abdrucken. Entscheidend ist das Prinzip:

> *Ein Vorteil zählt erst, wenn seine Qualität, seine Nebenwirkungen und sein eigener Aufwand sichtbar bleiben.*

Die genauen Messgrößen, Zeitfenster und statistischen Regeln gehören in das jeweilige Forschungspapier. Dort müssen sie so beschrieben werden, dass ein anderes Team den Versuch wiederholen oder kritisieren kann.

Wiederholung statt Vorführung

Ein einzelner schneller Lauf ist eine Begebenheit.

Rechner verändern sich. Caches sind warm oder kalt. Hintergrunddienste werden aktiv. Das Netzwerk ist für einige Sekunden ruhig. Ein Gerät wird heiß. Ein zufälliger Startwert begünstigt eine Variante.

Darum benötigt Forschung Wiederholung.

Die Reihenfolge der Versuche darf nicht unbemerkt nur einer Methode helfen. Störungen müssen beschrieben werden. Fehlgeschlagene Läufe dürfen nicht verschwinden. Ein Ergebnis muss auch dann sichtbar bleiben, wenn es die Erzählung des Buches schwächt.

Gerade darin unterscheidet sich Messung von Vision.

Die Vision fragt, was möglich sein könnte.

Die Messung fragt, ob es unter benannten Bedingungen wieder geschah.

Eine Grenze, keine vorweggenommene Antwort

Project Ant Colony besitzt zum Stand dieses Buches noch keine abgeschlossenen Hauptversuche.

Das ist keine Lücke, die mit Vermutungen gefüllt werden sollte. Es ist die Grenze zwischen diesem Buch und der anschließenden Forschung.

Das Buch legt offen, welche Art von Vorteil zählen dürfte und welche Abkürzungen verboten sind. Die Forschungspapiere werden später die technischen Einzelheiten tragen: Versuchsversionen, Messreihen, Baselines, Fehlerzustände und Resultate.

Damit kann das Buch leicht lesbar bleiben, ohne seine These gegen Widerspruch zu schützen.

Es öffnet die Tür. Es behauptet nicht, bereits hindurchgegangen zu sein.

Der nächste Schritt ist deshalb kein großes Siegesexperiment. Er ist eine Reihe kleiner Versuche, in denen auch ein Nein seinen Wert behält.

KAPITEL 21

Experimente und negative Ergebnisse

Die Forschung beginnt mit einem überprüfbaren Nein

Ein gestuftes Experimentprogramm behandelt auch ein überprüfbares Nein als wissenschaftlichen Fortschritt.

Ein neuer Forschungsansatz wirkt in seiner ersten Erzählung oft größer, als er in seinem ersten Experiment sein darf. Die Idee eines lernenden Meta-Betriebssystems kann bis zu ganzen Rechnerlandschaften reichen. Es kann Hardware beobachten, Verfahren auswählen, Aufgaben zerlegen und aus den Folgen seiner Entscheidungen lernen. Würde man all das zugleich bauen und prüfen, entstünde jedoch kein überzeugendes Experiment. Es entstünde ein schwer durchschaubares System, in dem ein Erfolg ebenso viele Ursachen haben könnte wie ein Scheitern.

Die Forschung muss deshalb kleiner beginnen als die Vision. Ihr erster Gegenstand ist nicht das fertige System, sondern eine begrenzte Entscheidung: Kann eine Maschine unter wechselnden Bedingungen zuverlässig zwischen mehreren bekannten Ausführungswegen wählen? Kann sie dabei besser abschneiden als eine einfache, gut gewählte Regel? Und lässt sich nachträglich verstehen, warum sie sich entschieden hat?

Diese Verkleinerung nimmt der Idee nichts. Sie macht sie angreifbar - und damit wissenschaftlich brauchbar.

Zuerst wird das Messinstrument geprüft

Bevor eine lernende Instanz beurteilt werden kann, muss die Umgebung verlässlich Auskunft geben. Das klingt selbstverständlich, ist es aber nicht. Zwei scheinbar gleiche Durchläufe können sich unterscheiden, weil ein anderer Prozess Speicher belegt, ein Gerät wärmer geworden ist, Daten noch in einem Zwischenspeicher liegen oder ein Treiber im Hintergrund eigene Entscheidungen trifft. Selbst eine einfache Zeitmessung enthält bereits eine kleine Geschichte des Systems.

Die erste Stufe der Experimente fragt deshalb noch nicht, ob das lernende Verfahren klug ist. Sie fragt, ob die Messung glaubwürdig ist. Wiederholt sich ein Ergebnis? Werden Störungen sichtbar? Können Anfangszustand, Ausführung und Ausgang

gemeinsam protokolliert werden? Stimmen verschiedene Messwege ausreichend überein?

Diese Arbeit ist unspektakulär. Gerade deshalb ist sie entscheidend. Ein lernendes System kann sonst nicht nur aus der Aufgabe lernen, sondern auch aus zufälligem Rauschen. Es würde Muster entdecken, die nur deshalb existieren, weil das Labor seine eigenen Schwankungen nicht kennt.

Dann kommen bekannte Wege

In der zweiten Stufe darf das System noch nichts erfinden. Es erhält mehrere bekannte Strategien, die alle dieselbe Aufgabe lösen können. Eine davon nutzt vielleicht einen starken Einzelknoten, eine andere verteilt Arbeit auf mehrere schwächere Geräte, eine dritte spart Energie und nimmt mehr Zeit in Kauf. Der lernende Teil soll lediglich einschätzen, welche Strategie unter den gerade beobachteten Bedingungen am ehesten zum Ziel führt.

Dieser Aufbau ist absichtlich bescheiden. Er trennt zwei Fragen, die sonst leicht ineinanderfallen: Ist die Auswahl gut - und ist die ausgewählte Strategie überhaupt gut implementiert? Wenn nur zwischen bereits geprüften Wegen gewählt wird, lässt sich der Wert der Entscheidung klarer erkennen.

Die wichtigste Vergleichsgröße ist dabei nicht eine naive Zufallsauswahl. Der faire Gegner ist eine einfache, sachkundige Regel. Sie könnte etwa immer das schnellste verfügbare Gerät wählen oder Aufgaben ab einer bestimmten Größe verteilen. Wenn das lernende System diese Regel nicht zuverlässig übertrifft, gibt es zunächst keinen Grund, seine zusätzliche Komplexität zu akzeptieren.

Erst danach wird die Welt unruhig

Eine Entscheidung ist nur dann interessant, wenn Bedingungen sich ändern. Die dritte Stufe führt deshalb Heterogenität und Störungen kontrolliert ein. Ein Knoten wird langsamer. Speicher wird knapp. Eine Verbindung verliert Bandbreite. Ein Gerät fällt zeitweise aus. Der Energieverbrauch erhält ein höheres Gewicht als die reine Geschwindigkeit. Nicht alles geschieht zugleich; jede Veränderung wird einzeln oder in kleinen, nachvollziehbaren Kombinationen geprüft.

Hier zeigt sich, ob das System nur eine Rangliste auswendig gelernt hat oder tatsächlich Zustände unterscheiden kann. Eine Strategie, die gestern richtig war, kann heute falsch sein. Genau in diesem Wechsel liegt die eigentliche Forschungsfrage. Der mögliche Vorteil entsteht nicht daraus, dass das System einen universell besten Weg kennt. Er entsteht daraus, dass es erkennt, wann ein anderer Weg sinnvoller geworden ist.

Auch diese Experimente brauchen Grenzen. Eine Störung darf nicht bloß behauptet, sondern muss beschrieben und wiederholbar erzeugt werden. Sonst verwandelt sich die vermeintliche Anpassungsfähigkeit in eine Erzählung, die jeder Beobachtung nachträglich eine passende Begründung gibt.

Neue Strategien kommen zuletzt

Erst wenn Auswahl, Messung und Reaktion auf Veränderungen getrennt verstanden sind, darf das System neue Kombinationen vorschlagen. Vielleicht zerlegt es eine Aufgabe anders, nutzt einen bislang übersehenen Speicherweg oder verbindet zwei unauffällige Geräte zu einer brauchbaren Ausführungsform. Das wäre der Punkt, an dem das Ameisenbild seine stärkste Bedeutung erhält: Nicht ein großer Entwurf wird von oben vorgegeben, sondern viele kleine Erkundungen hinterlassen Spuren, aus denen tragfähige Wege hervorgehen können.

Doch auch dann bleibt der Vorschlag vom Vollzug getrennt. Eine neue Strategie wird zunächst beschrieben, geprüft und in einer begrenzten Umgebung ausgeführt. Ein Fehler darf nicht ungefiltert zum nächsten Fehler führen. Das lernende Modell ist Erzeuger von Möglichkeiten, nicht letzte Instanz über das System.

Damit bleibt die Forschung zugleich offen und kontrollierbar. Sie kann überraschende Wege finden, ohne Sicherheit mit Kreativität zu verwechseln.

Das wichtige Ergebnis kann ein Scheitern sein

Bei einem solchen Projekt liegt die Versuchung nahe, nur nach einem positiven Beweis zu suchen. Doch mehrere negative Ergebnisse wären mindestens ebenso aufschlussreich.

Vielleicht schlägt eine einfache Regel das lernende System in fast allen Fällen. Dann wäre die zusätzliche Intelligenz an dieser Stelle nicht gerechtfertigt. Vielleicht enthält der beobachtete Systemzustand weniger nutzbare Information als angenommen. Dann müsste nicht das Modell größer, sondern die Messfrage besser werden. Vielleicht frisst die Suche nach einer guten Strategie mehr Zeit und Energie, als die Strategie später spart. Dann wäre der Optimierer selbst zum teuersten Teil des Problems geworden. Vielleicht erweist sich Heterogenität nicht als Ressource, sondern unter bestimmten Bedingungen nur als Verwaltungsaufwand. Auch das wäre eine klare Grenze der Idee.

Solche Befunde widerlegen nicht automatisch die gesamte Richtung. Sie bestimmen, wo sie trägt und wo nicht. Gerade ein Forschungsprogramm, das aus einer alternativen Technikgeschichte hervorgeht, braucht diese Nüchternheit. Die Gegenwart darf nicht nur deshalb passend erscheinen, weil man zuvor eine überzeugende andere Vergangenheit erzählt hat.

Vor dem Versuch steht das Versprechen

Jeder wesentliche Versuch sollte deshalb vor seinem Beginn festhalten, welche Aufgabe geprüft wird, welche Vergleichsverfahren gelten und woran Erfolg oder Misserfolg erkannt werden. Nicht jedes Detail muss für immer unveränderlich sein. Aber Änderungen müssen sichtbar bleiben. So lässt sich unterscheiden, ob eine Hypothese geprüft oder nach dem Ergebnis passend gemacht wurde.

Zu diesem Versprechen gehört auch, negative Resultate nicht aus der Geschichte zu entfernen. Wenn ein Ansatz langsamer, teurer oder instabiler ist, gehört dieser Befund in die Spur des Projekts. Er kann später verhindern, dass dieselbe Sackgasse unter einem neuen Namen noch einmal betreten wird.

Die technischen Papers werden dafür genaue Konfigurationen, Messreihen, Zufallsstarts, Fehlergrenzen und Auswertungsverfahren angeben. Dieses Buch tut etwas anderes. Es zeigt, warum diese Genauigkeit notwendig wird und welche Frage sie schützen soll. Es behauptet noch kein Ergebnis.

Eine Forschung, die ihre Spuren behält

Das Ant-Colony-Motiv beschreibt damit nicht nur die geplante Technik. Es beschreibt auch eine Form des Forschens. Jeder Versuch hinterlässt eine Spur: den beobachteten Zustand, die getroffene Entscheidung, die Ausführung und ihre Folgen. Gute Wege werden sichtbarer. Schlechte Wege verschwinden nicht vollständig, sondern bleiben als begründete Warnung erhalten.

So kann aus einer großen Vermutung ein prüfbares Programm entstehen. Es beginnt nicht mit dem Versprechen einer neuen Rechnerordnung. Es beginnt mit einer kleinen Entscheidung, einem fairen Vergleich und der Bereitschaft, ein klares Nein als Fortschritt anzuerkennen.

Genau an diesem Punkt endet die Aufgabe des Buches. Die nächste Bewegung gehört dem Labor.

KAPITEL 22

Was die alternative Evolution prüfbar macht

Rückkehr zur Abzweigung

Der historische Abzweig öffnet eine Forschungsrichtung, ohne ihre späteren technischen Antworten vorwegzunehmen.

Am Anfang dieses Buches stand eine einfache Verschiebung der Perspektive. Die Geschichte der künstlichen Intelligenz wurde nicht nur als Geschichte immer besserer Modelle betrachtet. Sie erschien als Folge von Brüchen, in denen mathematische Ideen, verfügbare Daten, Rechenhardware, Software und gesellschaftliche Erwartungen einander verstärkten. CUDA war in dieser Geschichte nicht die Ursache von KI. Es war ein besonders wirksamer Übergang: eine Verbindung zwischen programmierbarer Parallelität und einem Ökosystem, das diese Parallelität immer leichter nutzbar machte.

Dann folgte die Gegenfrage: Was wäre geschehen, wenn genau diese Verbindung nicht in derselben Form entstanden wäre?

Eine solche Frage kann keine verlorene Vergangenheit beweisen. Es gibt kein zweites Archiv einer Welt ohne CUDA, in dem wir nachsehen könnten. Die alternative Evolution wird erst dann wissenschaftlich interessant, wenn sie nicht als Behauptung über das Gewesene behandelt wird, sondern als Werkzeug für die Gegenwart. Sie muss zeigen, welche Möglichkeiten durch den dominanten Weg weniger sichtbar wurden - und wie sich wenigstens einige davon heute prüfen lassen.

Vier Bedingungen für ein brauchbares Gegenbild

Die erste Bedingung sind reale Anker. Eine alternative Entwicklung darf nicht aus beliebigen Wünschen bestehen. CPU-Cluster, offene Zwischensprachen, portable Programmiermodelle, FPGAs, spezialisierte Beschleuniger und heterogene Laufzeitsysteme haben tatsächlich existiert oder existieren noch. Sie beweisen nicht, wie die gesamte Geschichte anders verlaufen wäre. Aber sie zeigen, dass andere technische Bausteine vorhanden waren.

Die zweite Bedingung sind konkurrierende Welten. Ohne CUDA wäre nicht automatisch ein einziges offenes und vielfältiges Paradies entstanden. Ebenso denkbar wären starke vertikale Systeme einzelner Anbieter oder eine fragmentierte Landschaft schwer

vereinbarer Werkzeuge gewesen. Das Gegenbild bleibt glaubwürdig, wenn es auch unbequeme Möglichkeiten zulässt.

Die dritte Bedingung ist eine Übersetzung in die Gegenwart. Aus dem Gedankenexperiment muss eine Frage werden, die an realer Hardware, realer Software und unter endlichen Kosten geprüft werden kann. Genau hier entsteht der Übergang zum Meta-Betriebssystem und zum Decision Compiler. Beide behandeln die vorhandene Rechnerlandschaft nicht als lästigen Rest, sondern als veränderliche Menge von Fähigkeiten.

Die vierte Bedingung ist ein erlaubtes Scheitern. Wenn jedes Ergebnis nachträglich als Bestätigung der alternativen Geschichte gedeutet werden kann, ist nichts gewonnen. Die Idee muss an einer einfachen Regel, an ihrem eigenen Verwaltungsaufwand oder an der tatsächlichen Bedeutung von Heterogenität scheitern dürfen.

Erst zusammen machen diese Bedingungen aus einer kontrafaktischen Erzählung ein Forschungsinstrument.

Die Bruchfolge reicht tiefer als bis zum Beschleuniger

Das neue Zwischenspiel dieses Buches hat den Abzweig deshalb weiter verfolgt. Eine andere Beschleunigergeschichte hätte nicht nur beeinflusst, welche Chips gekauft werden. Sie hätte auch verändert, wie Rechenkerne beschrieben, wie Speicher sichtbar gemacht, wie Betriebssysteme Arbeit verteilt und wie Anwendungen Fähigkeiten angefordert hätten.

Das Wort Kernel berührt dabei zwei Ebenen. Auf der einen bezeichnet es den kleinen Rechenkern, der eine konkrete Operation auf Daten ausführt. Auf der anderen bezeichnet es den innersten Teil des Betriebssystems, der Prozessorzeit, Speicher und Geräte ordnet. Zwischen beiden Ebenen liegen Compiler, Laufzeiten, Treiber und Bibliotheken. Sie entscheiden, ob Unterschiede der Hardware hinter einer einheitlichen Oberfläche verschwinden oder als gestaltbare Eigenschaften erhalten bleiben.

Auch Anwendungen wären Teil dieser Entwicklung gewesen. In einer stärker heterogenen Welt hätten sie möglicherweise seltener ein bestimmtes Gerät vorausgesetzt und häufiger beschrieben, was sie benötigen: eine Speichergröße, eine Genauigkeit, eine Antwortzeit, einen Energieverbrauch oder eine Form der Vertraulichkeit. Software wäre dadurch nicht automatisch einfacher geworden. Viel Aufwand hätte sich nur verschoben - weg von einer dominanten Plattform und hin zu Übersetzung, Beobachtung und Aushandlung.

Dieser Gedanke ist für Ant Colony zentral. Er zeigt, dass der Forschungsansatz nicht erst bei der Verteilung fertiger KI-Aufgaben beginnt. Er berührt eine ganze Kette von Entscheidungen zwischen Anwendung und Maschine. Dennoch muss nicht die gesamte Kette zugleich neu gebaut werden. Ein erster Versuch darf an einer einzigen, sauber begrenzten Stelle ansetzen.

Von der anderen Geschichte zur übersehenen Möglichkeit

Der dominante Entwicklungspfad hat eine starke Frage begünstigt: Wie lässt sich mehr Rechenleistung in eine möglichst gleichförmige Trainingsmaschine verwandeln? Diese Frage war erfolgreich. Sie führte zu leistungsfähigen Modellen, stabilen Werkzeugketten und einer erstaunlichen Beschleunigung des Feldes.

Die alternative Linie stellt daneben eine andere Frage: Wie lässt sich eine ungleichförmige Landschaft so beobachten und organisieren, dass ihre Unterschiede nutzbar werden?

Das ist kein romantischer Gegensatz zwischen großen und kleinen Computern. Auch ein heterogenes System kann verschwenderisch, geschlossen oder schlecht kontrollierbar sein. Und ein hoch integriertes Beschleunigersystem kann für eine Aufgabe eindeutig die beste Lösung darstellen. Die Forschung sucht daher nicht nach dem moralisch besseren Chip. Sie untersucht, ob algorithmische Verbesserung dort Handlungsspielraum erzeugen kann, wo das heutige Narrativ vor allem einen Mangel an Beschleunigern sieht.

Wird dieser Spielraum gefunden, verändert sich die Bedeutung vorhandener Hardware. Ein alter Rechner ist dann nicht bloß eine langsame Version des aktuellen Standards. Er kann eine bestimmte Speicherlage, Energiecharakteristik, Verfügbarkeit oder Nähe zu Daten anbieten. Mehrere solcher Eigenschaften können gemeinsam einen brauchbaren Weg bilden. Genau darin liegt das evolutionäre Motiv: Nicht das stärkste einzelne Element entscheidet zwangsläufig, sondern die Passung zwischen Variation, Umgebung und Auswahl.

Zwei ehrliche Ausgänge

Die Experimente können zeigen, dass dieser Gedanke nur selten trägt. Vielleicht bleiben Übersetzung, Messung und Koordination zu teuer. Vielleicht ist die spezialisierte Beschleunigerstraße bei fast allen relevanten Aufgaben nicht nur bequemer, sondern auch unter Einbezug ihrer Nebenkosten überlegen. Dann hätte das Projekt eine Grenze sichtbar gemacht. Es würde erklären, warum die Monokultur nicht allein aus Gewohnheit fortbesteht.

Sie können aber auch zeigen, dass Nebenwege unter bestimmten Bedingungen belastbar werden: bei knapper Energie, bei lokal verbleibenden Daten, bei unregelmäßiger Verfügbarkeit, bei bereits vorhandener Hardware oder bei Aufgaben, die nicht die größte mögliche Modellskala benötigen. Dann wäre nicht bewiesen, dass die Geschichte ohne CUDA besser verlaufen wäre. Bewiesen wäre etwas Nützlicheres: dass eine von dieser Gegenwart verdeckte Entwurfsrichtung heute messbaren Wert besitzt.

Beide Ausgänge sind wissenschaftlich ergiebig. Sie ersetzen Spekulation durch eine Landkarte von Bedingungen.

Eine andere Vorstellung von Fortschritt

Gesellschaftlich führt die alternative Evolution zu einer weiteren Frage. Wenn Fortschritt vor allem als Wachstum von Modell, Rechenzentrum und Kapitalbedarf erscheint, werden viele andere Verbesserungen leicht als bloße Sparmaßnahmen gelesen. Ein Verfahren, das dieselbe Aufgabe mit weniger neuer Hardware, größerer lokaler Kontrolle oder längerer Nutzung vorhandener Geräte löst, gilt dann nicht als neue Leistung, sondern als Annäherung an den eigentlichen Standard.

Ant Colony schlägt eine andere Bewertung vor. Algorithmische Verbesserung kann selbst eine Form von Infrastruktur sein. Sie kann Zugänge erweitern, Abhängigkeiten verändern und den Wert dessen neu bestimmen, was bereits vorhanden ist. Damit wird Effizienz nicht automatisch zur Tugend und Dezentralität nicht automatisch zur Gerechtigkeit. Aber beide werden zu Fragen, die technisch und gesellschaftlich gemeinsam untersucht werden können.

Das ist die Verbindung zu den früheren Brüchen. Neue Technik verändert nicht nur Möglichkeiten. Sie verändert auch, was eine Zeit für vernünftig, modern und unvermeidlich hält. Eine alternative Evolutionsvorstellung macht diese Urteile wieder sichtbar.

Hier öffnet sich die Forschung

Dieses Buch endet deshalb nicht mit einer fertigen Architektur und nicht mit einem versprochenen Durchbruch. Es endet mit einer präziseren Öffnung.

Die historische Untersuchung hat gezeigt, wie sich ein dominanter Pfad aus vielen Verstärkungen bilden konnte. Das Gegenbild hat mehrere andere Welten entworfen, ohne eine davon zur verlorenen Wahrheit zu erklären. Das Zwischenspiel hat den Abzweig bis in Rechenkerne, Betriebssysteme und Anwendungen verfolgt. Der Ant-Colony-Ansatz hat daraus eine gegenwärtige Forschungsfrage gewonnen: Kann ein beobachtendes, lernendes und kontrollierbares System heterogene Ressourcen unter wechselnden Bedingungen besser nutzen als einfache, feste Regeln?

Die folgenden Papers müssen diese Frage in technischer Tiefe beantworten. Dort gehören formale Modelle, Systemgrenzen, Versuchsaufbauten, Messreihen, Fehleranalysen und reproduzierbare Artefakte hin. Das Buch hat ihnen nicht vorgegriffen. Es hat den Weg bis zu ihrer Schwelle freigelegt.

Vielleicht erweist sich der Beschleuniger-Highway als der tragfähigste Weg. Vielleicht finden kleine Erkundungen Pfade, die bisher unter seinem Lärm kaum zu sehen waren. In beiden Fällen beginnt Erkenntnis dort, wo die Erzählung eine prüfbare Entscheidung wird.

Literatur und Quellen

Historische Grundlagen

- Donald O. Hebb: *The Organization of Behavior: A Neuropsychological Theory*. John Wiley and Sons, 1949.
- Warren S. McCulloch und Walter Pitts: „A Logical Calculus of the Ideas Immanent in Nervous Activity“. *The Bulletin of Mathematical Biophysics* 5 (1943), 115-133. DOI: 10.1007/BF02478259.
- Marvin Minsky und Seymour A. Papert: *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1969.
- Mikel Olazaran: „A Sociological Study of the Official History of the Perceptrons Controversy“. *Social Studies of Science* 26, Nr. 3 (1996), 611-659. DOI: 10.1177/030631296026003005.
- Frank Rosenblatt: „The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain“. *Psychological Review* 65, Nr. 6 (1958), 386-408. DOI: 10.1037/h0042519.
- David E. Rumelhart, Geoffrey E. Hinton und Ronald J. Williams: „Learning Representations by Back-Propagating Errors“. *Nature* 323 (1986), 533-536. DOI: 10.1038/323533a0.
- Paul J. Werbos: *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Dissertation, Harvard University, 1974.

Zeitgeschichtliche und kulturelle Kontexte

- American Museum of Natural History: „Revisiting Asimov's Three Laws of Robotics“, zur Veröffentlichung von *Runaround* 1942 und *I, Robot* 1950.
- NASA History: „Sputnik and the Dawn of the Space Age“, zum Start von Sputnik 1 am 4. Oktober 1957 und zum Beginn des Raumfahrtzeitalters.
- National Museum of American History: „Science and Political Protest“, zum Forschungsstreik am MIT am 4. März 1969.
- Smithsonian Institution: „1969: A Year in the Collections“, zu Apollo 11, Woodstock, sozialen Konflikten und Antikriegsprotesten.
- Westdeutscher Rundfunk: „8. September 1961 - Erstes Perry-Rhodan-Heft erscheint“, zeitgeschichtlicher Rückblick.

Daten, Beschleunigung und Skalierung

- Rishi Bommasani et al.: *On the Opportunities and Risks of Foundation Models*. Stanford Center for Research on Foundation Models, 2021. arXiv:2108.07258.

- Tom B. Brown et al.: „Language Models are Few-Shot Learners“. *Advances in Neural Information Processing Systems 33* (2020), 1877-1901.
- Ian Buck et al.: „Brook for GPUs: Stream Computing on Graphics Hardware“. *ACM Transactions on Graphics* 23, Nr. 3 (2004), 777-786. DOI: 10.1145/1015706.1015800.
- Jeffrey Dean et al.: „Large Scale Distributed Deep Networks“. *Advances in Neural Information Processing Systems 25* (2012), 1223-1231.
- Jia Deng et al.: „ImageNet: A Large-Scale Hierarchical Image Database“. *IEEE Conference on Computer Vision and Pattern Recognition* (2009), 248-255. DOI: 10.1109/CVPR.2009.5206848.
- Jordan Hoffmann et al.: „Training Compute-Optimal Large Language Models“. arXiv:2203.15556, 2022.
- Jared Kaplan et al.: „Scaling Laws for Neural Language Models“. arXiv:2001.08361, 2020.
- Alex Krizhevsky, Ilya Sutskever und Geoffrey E. Hinton: „ImageNet Classification with Deep Convolutional Neural Networks“. *Advances in Neural Information Processing Systems 25* (2012), 1097-1105.
- NVIDIA: *CUDA Programming Guide*, historische Einführung zu CUDA, 2006.
- NVIDIA: „Accelerate Machine Learning with the cuDNN Deep Neural Network Library“, 2014.
- Ashish Vaswani et al.: „Attention Is All You Need“. *Advances in Neural Information Processing Systems 30* (2017), 5998-6008.

Alternative Hardware- und Modellpfade

- Filipp Akopyan et al.: „TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip“. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34, Nr. 10 (2015), 1537-1557. DOI: 10.1109/TCAD.2015.2474396.
- Eric Chung et al.: „Serving DNNs in Real Time at Datacenter Scale with Project Brainwave“. *IEEE Micro* 38, Nr. 2 (2018), 8-20.
- William Fedus, Barret Zoph und Noam Shazeer: „Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity“. *Journal of Machine Learning Research* 23, Nr. 120 (2022), 1-39.
- Song Han, Huizi Mao und William J. Dally: „Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding“. *International Conference on Learning Representations*, 2016.
- Norman P. Jouppi et al.: „In-Datacenter Performance Analysis of a Tensor Processing Unit“. *44th International Symposium on Computer Architecture* (2017), 1-12.
- Khronos Group: „OpenCL 1.0 Released“, 8. Dezember 2008.
- Patrick Lewis et al.: „Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks“. *Advances in Neural Information Processing Systems 33* (2020), 9459-9474.

Compiler, Planung und Algorithmussuche

- Jason Ansel et al.: „OpenTuner: An Extensible Framework for Program Autotuning“. *23rd International Conference on Parallel Architectures and Compilation Techniques* (2014), 303-316. DOI: 10.1145/2628071.2628092.
- Tianqi Chen et al.: „TVM: An Automated End-to-End Optimizing Compiler for Deep Learning“. *13th USENIX Symposium on Operating Systems Design and Implementation* (2018), 578-594.
- Alhussein Fawzi et al.: „Discovering Faster Matrix Multiplication Algorithms with Reinforcement Learning“. *Nature* 610 (2022), 47-53.
- LLVM Project: „Machine Learning - Guided Optimization (MLGO)“, offizielle LLVM-Dokumentation.
- Hongzi Mao et al.: „Learning Scheduling Algorithms for Data Processing Clusters“. *ACM SIGCOMM* (2019), 270-288.
- Daniel J. Mankowitz et al.: „Faster Sorting Algorithms Discovered Using Deep Reinforcement Learning“. *Nature* 618 (2023), 257-263.
- Esteban Real et al.: „AutoML-Zero: Evolving Machine Learning Algorithms From Scratch“. *Proceedings of the 37th International Conference on Machine Learning* (2020), 8007-8019.
- Wencong Xiao et al.: „Gandiva: Introspective Cluster Scheduling for Deep Learning“. *13th USENIX Symposium on Operating Systems Design and Implementation* (2018), 595-610.
- Lianmin Zheng et al.: „Ansor: Generating High-Performance Tensor Programs for Deep Learning“. *14th USENIX Symposium on Operating Systems Design and Implementation* (2020), 863-879.

Heterogene Software und Betriebssysteme

- Heterogeneous System Architecture Foundation: *HSA Platform System Architecture Specification*, offizielle Systemspezifikation.
- Khronos Group: „SPIR-V - The Industry Open Standard Intermediate Language for Parallel Compute and Graphics“, offizielle Standardübersicht.
- Khronos Group: *SYCL 2020 Specification*, offene Spezifikation für heterogene Programmierung.
- Linux Kernel Documentation: „Capacity Aware Scheduling“, offizielle Dokumentation zur Planung auf Prozessoren mit ungleicher Kapazität.
- Linux Kernel Documentation: „Heterogeneous Memory Management“, offizielle Dokumentation zur Einbindung von Gerätespeicher in Kernel-Speicherpfade.
- LLVM Project: „Multi-Level Intermediate Representation Overview“, offizielle Projektdokumentation.

Kommunikation und Kolonie

- Marco Dorigo und Luca Maria Gambardella: „Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem“. *IEEE Transactions on Evolutionary Computation* 1, Nr. 1 (1997), 53-66. DOI: 10.1109/4235.585892.
- Yujun Lin et al.: „Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training“. *International Conference on Learning Representations*, 2018.
- Sebastian U. Stich: „Local SGD Converges Fast and Communicates Little“. *International Conference on Learning Representations*, 2019.

Lokale Forschungsartefakte

- Project Ant Colony: *Hardware-Inventur und Versuchstauglichkeit*, sanitierter Live-Snapshot vom 14. Juli 2026.
- Project Ant Colony: *Forschungsstand, Neuheitsgrenze und Beitrag*, Forschungsnotiz 2026.
- Project Ant Colony: *Messung, Vorregistrierung und Falsifikation*, Forschungsnotiz 2026.
- Project Ant Colony: *Zyklus 0 - Messsystem validieren*, Arbeitsprotokoll 2026.

Die vollständigen maschinenlesbaren Angaben und Direktlinks stehen in `03_literature/bibliography.bib` und `03_literature/02_source_matrix.md`.

Über den Autor

Joachim Müller-Peter

Joachim Müller-Peter arbeitet an der Schnittstelle von gewachsenen KMU-Systemen, Datenbanken, Web-Anwendungen, FileMaker, lokaler Infrastruktur und künstlicher Intelligenz. Sein Schwerpunkt ist die Frage, wie bestehende Systeme für Menschen und KI-Agenten verständlich, kontrollierbar, überprüfbar und rückbaubar werden.

Aus der praktischen Arbeit mit realen Betriebsumgebungen entstand sein Interesse an einer KI, die nicht nur als Modell, sondern als vollständiges System betrachtet wird: mit Energiebedarf, Hardware, Netzen, Speicher, Rollen, Regeln und Verantwortung.

Mit Model OS und Project Ant Colony untersucht er, wie lokale Rechner, spezialisierte Experten, deterministische Kontrollschichten und Frontier-Modelle sinnvoll zusammenspielen können. *Der Körper der KI* ordnet diese technische Arbeit in einen größeren wirtschaftlichen und gesellschaftlichen Zusammenhang ein.

Joachim Müller-Peter ist Gründer von itbois in der Schweiz.

> *„Algorithmische Verbesserung führt zu bisher übersehenen Möglichkeiten.“*

itbois

https://next.itbois.ch/forschung/

Von diesem Buch zur Forschung

Dieses Buch enthält alle 22 Kapitel und das Zwischenspiel der Bucharchitektur. Historische Tatsachen, vorsichtige Synthesen und kontrafaktische Szenarien bleiben getrennt. Quellenhinweise stehen kapitelweise sowie im anschließenden Literaturverzeichnis.

Diese Buchfassung eröffnet die Forschung und nimmt ihre Ergebnisse nicht vorweg. Vollständige Versuchsspezifikationen, Messreihen, formale Modelle und reproduzierbare Artefakte erscheinen in separaten Forschungspapieren. Das Buch bewahrt die historische, gesellschaftliche und begriffliche Bewegung, aus der diese Arbeiten hervorgehen.

Leitthese: *Algorithmische Verbesserung führt zu bisher übersehenen Möglichkeiten.* Ob diese These den realen Vergleich besteht, bleibt Aufgabe der anschließenden Forschung.