**Was ist Whisper?**

**Whisper** ist ein Open-Source-Spracherkennungsmodell (ASR – Automatic Speech Recognition), das von **OpenAI** entwickelt wurde. Es kann gesprochene Sprache aus Audiodateien erkennen und in Text umwandeln („transkribieren“).

 **Ablauf:**

1. **Audio wird in kleine Stücke geschnitten** und in ein passendes Format (z.B. Spektrogramm) umgewandelt.
2. **Das Modell analysiert diese Stücke**, erkennt Sprache, Wörter und Satzstrukturen.
3. **Ergebnis:** Das Modell gibt den erkannten Text zurück – optional auch mit Zeitmarken (wann wurde was gesagt).

 **Multilingual:**  
Whisper kann viele Sprachen erkennen, übersetzen und transkribieren.

**Wer hat Whisper gemacht?**

* **Entwickler:**  
  **OpenAI** (bekannt durch ChatGPT, DALL·E etc.)
* **Veröffentlichung:**  
  September 2022, als Open-Source auf GitHub.
* **Lizenz:**  
  MIT-Lizenz, frei nutzbar.

Whisper ist ein System zur automatischen Spracherkennung (ASR), das mit 680.000 Stunden mehrsprachiger und multitaskingfähiger überwachter Daten aus dem Internet trainiert wurde.

**1. Überwachte Daten (supervised data)**

„Überwacht“ bedeutet im Machine Learning:

* Die Trainingsdaten bestehen aus **Eingaben** (z. B. Audiodateien) und den **dazugehörigen Ausgaben/Zielen** (z. B. die Transkription als Text).
* Das Modell lernt, **wie es von der Eingabe zur richtigen Ausgabe kommt**, indem es an vielen Beispielen trainiert wird.

Beispiel: Eine Audiodatei mit dem Satz *„Guten Morgen.“* + das passende Transkript *„Guten Morgen.“*

**2. Multitaskingfähig (multitask-capable)**

„Multitasking“ bedeutet hier, dass Whisper nicht nur **eine einzige Aufgabe** lernt, sondern **mehrere verwandte Aufgaben gleichzeitig**, z. B.:

|  |  |
| --- | --- |
| **Aufgabe** | **Beschreibung** |
| **Spracherkennung (ASR)** | Audio → Text (in Originalsprache) |
| **Spracherkennung mehrsprachig** | ASR in vielen Sprachen |
| **Spracherkennung + Sprache erkennen** | Modell erkennt auch, **welche Sprache** gesprochen wird |
| **Übersetzung** | Audio in Fremdsprache → **Transkript in Englisch** |

Das Forschungsteam von Whisper zeigt, dass die Verwendung eines so großen und vielfältigen Datensatzes zu einer verbesserten Robustheit hinsichtlich Akzenten, Hintergrundgeräuschen und Fachsprache führt. Darüber hinaus ermöglicht Whisper die Transkription in mehrere Sprachen sowie die Übersetzung aus diesen Sprachen ins Englische. Whisper/ Open AI stellt den **Inferenzcode1** als Open Source zur Verfügung, der als Grundlage für die Entwicklung nützlicher Anwendungen und für die weitere Forschung zur robusten Sprachverarbeitung dienen können.

*1Inferenzcode bezeichnet den Teil eines Programms, der ein trainiertes Machine-Learning-Modell verwendet, um neue Daten zu verarbeiten – also Vorhersagen oder Ergebnisse zu erzeugen, ohne das Modell neu zu trainieren.*

*Ein Bild, das Text, Diagramm, Screenshot, Plan enthält.

KI-generierte Inhalte können fehlerhaft sein.*

1. **Eingabe (Links unten)**:  
   Die Architektur von Whisper ist ein einfacher, durchgehender Ansatz, der als **Encoder-Decoder-Transformator** implementiert ist. Das eingegebene Audiomaterial wird in 30-Sekunden-Blöcke aufgeteilt, in ein Log-Mel-Spektrogramm umgewandelt und dann an einen Encoder übergeben. Ein Decoder wird darauf trainiert, die entsprechenden Untertitel vorherzusagen, vermischt mit speziellen Token, die das einzelne Modell anweisen, Aufgaben wie Spracherkennung, Zeitstempel auf Wortverbindungsebene, mehrsprachige Sprachtranskription und Sprachübersetzung ins Englische auszuführen.

Der Encoder-Decoder-Transformator ist ein neuronales Netzwerk-Architekturmodell. Beatandteile des Encoders :

1. **Self Attention Mechanismus**

Der Self-Attention-Mechanismus sorgt dafür, dass das Modell für jedes Wort in einer Sequenz bestimmen kann, auf welche anderen Wörter es „achten“ soll, um eine möglichst gute Repräsentation zu bilden.

**Wie funktioniert das?**

* Jedes Wort wird in einen Vektor (eine Zahlenreihe) umgewandelt.
* Für jedes Wort werden drei Vektoren berechnet: Query (Frage), Key (Schlüssel), Value (Wert).
* Die Query eines Wortes wird mit den Keys aller Wörter verglichen (über ein Skalarprodukt), um zu berechnen, wie sehr die einzelnen Wörter zueinander „passen“ (dies ergibt die Attention Scores).
* Mit diesen Scores werden die Value-Vektoren aller Wörter gewichtet und zu einem neuen Vektor für das betrachtete Wort zusammengefasst.
* Dadurch kann jedes Wort Informationen aus der gesamten Sequenz beziehen, nicht nur von seinen direkten Nachbarn.

Stell dir vor, du liest einen Satz:  
„Die Katze jagt die Maus, weil sie hungrig ist.“

Du möchtest wissen, auf wen sich „sie“ bezieht. Ist es die Katze oder die Maus?  
Der Self-Attention-Mechanismus hilft dem Computer, solche Zusammenhänge zu erkennen.

**Wie funktioniert das?**  
Jedes Wort im Satz „schaut“ auf die anderen Wörter und entscheidet, welche besonders wichtig sind.  
So kann das Modell verstehen, dass „sie“ sich in diesem Fall auf „die Katze“ bezieht.

**+-----+ +-----+ +-----+ +-----+ +-----+**

**| Die | ---> | Katze | <-- | jagt | --> | die | <--- | Maus |**

**+-----+ +-----+ +-----+ +-----+ +-----+**

**\ ^ \ ^ / / /**

**\\_\_\_\_\_\_\_|\_\_\_\_\_\_\\_\_\_\_\_|\_\_\_\_/\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_/**

**^ \ / /**

**| \ / /**

**+--------\------/------+**

Jedes Wort „achtet“ auf andere Wörter im Satz (Pfeile zeigen die Aufmerksamkeit).

**Was bedeutet „2x Conv1D + GELU“?**

* **2x Conv1D**  
  Das bedeutet, dass zwei aufeinanderfolgende eindimensionale Faltungsschichten (1D-Convolutional-Layer) auf das Log-Mel-Spektrogramm angewandt werden.
  + **Conv1D (1D-Convolution)**:  
    Dies ist eine Filteroperation, die entlang einer Achse (hier: Zeitachse des Audios) läuft. Sie erkennt lokale Muster, z.B. kurze Tonfolgen oder Sprachlaute.
  + **Warum zwei Schichten?**  
    Durch mehrere Schichten kann das Modell zunächst sehr lokale Muster erkennen (z.B. einzelne Silben) und dann in der zweiten Schicht schon komplexere Zusammenhänge (z.B. Silbenfolgen oder Wortanfänge).
* **GELU**  
  Nach jeder Conv1D-Schicht folgt eine sogenannte GELU-Aktivierungsfunktion (Gaussian Error Linear Unit).
  + Sie sorgt dafür, dass das Modell nicht nur lineare Zusammenhänge lernt, sondern auch komplexere, nichtlineare Muster erkennt.

-----

**1. Self-Attention-Mechanismus**

Der Self-Attention-Mechanismus sorgt dafür, dass das Modell für jedes Wort in einer Sequenz bestimmen kann, auf welche anderen Wörter es „achten“ soll, um eine möglichst gute Repräsentation zu bilden.

**Wie funktioniert das?**

* Jedes Wort wird in einen Vektor (eine Zahlenreihe) umgewandelt.
* Für jedes Wort werden drei Vektoren berechnet: Query (Frage), Key (Schlüssel), Value (Wert).
* Die Query eines Wortes wird mit den Keys aller Wörter verglichen (über ein Skalarprodukt), um zu berechnen, wie sehr die einzelnen Wörter zueinander „passen“ (dies ergibt die Attention Scores).
* Mit diesen Scores werden die Value-Vektoren aller Wörter gewichtet und zu einem neuen Vektor für das betrachtete Wort zusammengefasst.
* Dadurch kann jedes Wort Informationen aus der gesamten Sequenz beziehen, nicht nur von seinen direkten Nachbarn.

**Beispiel:**  
Im Satz „Die Katze jagt die Maus“ kann das Wort „jagt“ sowohl auf „Katze“ (Subjekt) als auch auf „Maus“ (Objekt) achten, um die Bedeutung besser zu verstehen.

1. **Feedforward-Netzwerk**

Es ermöglicht eine tiefere Verarbeitung der von der Attention erfassten Informationen.Die ursprüngliche Bedeutung wird **verfeinert**, je nachdem, wo das Wort steht und was drumherum gesagt wird.

**Beispiel:**

Satz:

„Der Hund rennt über die Wiese.“

Dann bekommt „Hund“ eine Bedeutung wie: „Das ist ein Tier, das gerade etwas tut.“

Aber im Satz:

„Ich habe einen Hund gekauft.“

Dann bekommt „Hund“ eher die Bedeutun „Ein neues Haustier, das jemand bekommt.“

**Was macht das Feedforward-Netzwerk?**

Stell dir vor, das Feedforward-Netzwerk ist wie ein kleiner **Koch**, der aus einem Rezept (dem Wort-Steckbrief) etwas Neues macht.

Beispiel:

* Der Steckbrief von „Hund“ ist wie eine **Liste von Zutaten**:

„etwas Tierisches, aktiv, freundlich …“

Das Feedforward-Netzwerk hat gelernt:

* Wenn die Zutaten so und so aussehen,
* dann mach daraus einen neuen Steckbrief – z. B. einen, der besser zur Handlung passt („Hund rennt“, „Hund bellt“, …).

**🔍 Wie lernt das Feedforward-Netzwerk das?**

**1. Zufällig starten**

Am Anfang kann das Netzwerk nichts – es macht nur **Zufallsentscheidungen**.

**2. Training mit vielen Texten**

Dann wird das Netzwerk mit **vielen Millionen echten Texten** trainiert.

Beispiel:

* Es bekommt den Satz:

„Der Hund rennt über die Wiese.“

* Und soll das nächste Wort **vorhersagen**: z. B. „über“.

Wenn das Feedforward-Netzwerk schlechte Werte liefert (falscher Vorschlag), sagt der Computer:

❌ „Das war falsch, versuch’s beim nächsten Mal besser.“

Und dann **passt sich das Netzwerk an**:

* Es verändert die „Gewichte“ im Inneren – das sind kleine Stellschrauben.
* Dadurch wird das Ergebnis beim nächsten Mal **etwas besser**.

Das passiert **immer wieder**, millionenfach.

**⚙️ Was passiert im Inneren?**

Ganz grob gesagt:

1. Der Wort-Steckbrief (z. B. „Hund“) wird **reingeschoben**.
2. Das Netzwerk **rechnet** damit – wie ein Rezept:  
   „Zutat A mal so viel, Zutat B mal so viel … und alles zusammenmixen.“
3. Es kommt ein **neuer Steckbrief** raus – ein verfeinertes Wortverständnis.

Das Netzwerk verändert dabei **nicht das Wort selbst**, sondern **dessen Bedeutung im Satz**.

**🧩 Fazit (einfach gesagt):**

* Das Feedforward-Netzwerk bekommt eine „Zahlen-Bedeutung“ eines Wortes.
* Es wurde durch Training mit vielen Texten **dazu gebracht**, sinnvolle Bedeutungen auszugeben.
* Es rechnet mit gelernten Regeln (wie ein Koch mit einem Rezept).
* Je nachdem, was das Netzwerk gelernt hat, gibt es eine neue, passende Bedeutung für das Wort zurück.

**3. Kontextvektoren**

**Erklärung:**  
Jedes Wort bekommt nach diesen beiden Schritten eine Art „Gedächtnis“ – ein Zahlenpaket, das all das Wissen über das Wort und seine Nachbarn speichert.  
Das nennt man „Kontextvektor“. Damit kann der Computer z. B. unterscheiden, ob „Bank“ eine Sitzbank oder ein Geldinstitut meint – je nachdem, welche Wörter drum herum stehen.

**Zusammengefasst Encoder Decoder - Transformator**

* **Self-Attention:** Jedes Wort achtet darauf, was andere Wörter bedeuten und welche wichtig sind.
* **Feedforward-Netzwerk:** Überlegt für jedes Wort einzeln, was es mit der neuen Bedeutung macht.
* **Kontextvektoren:** Sind wie ein Gedächtnis für jedes Wort, das den Zusammenhang festhält.

**DECODER**

**Schritt 3: Erkennen der Laute und Wörter (mit neuronalen Netzen)**

Ein trainiertes Modell (z.B. ein Transformer-Encoder) schaut sich das Spektrogramm an und erkennt darin Muster:

* Welche Abschnitte entsprechen welchen Lauten (z.B. „g“, „u“, „t“, „en“)?
* Wie werden die Laute zu Wörtern zusammengesetzt?

Das Modell gibt für jeden Abschnitt des Audios die wahrscheinlichsten Laute/Buchstaben/Wörter aus.

Die Transformer-Encoder wurden mit Audiofiles und den dazugehörigen richtigen Textfiles trainiert. Spezielle – Aligner Programme verbinden das Textfile mit dem Audio Input damit der Transformer-Encoder sinnvoll Trainiert werden kann.

**Startsignal**

* Der Decoder bekommt als Startzeichen sogenannte Steuer-Tokens (z.B. SOT = Start Of Transcript, Sprache, Modus).
* Damit weiß er: „Jetzt soll ich Text ausgeben!“

**2. Eingabe der bisherigen Tokens**

* Anfangs ist das nur das Startsignal.
* Sobald erste Wörter erzeugt wurden, nimmt er zusätzlich auch die schon generierten Wörter als Input.

**3. Cross-Attention auf Encoder-Ausgabe**

* Der Decoder schaut bei jedem Schritt auf die „Zusammenfassung“ des Audios, die vom Encoder kommt.
* Über Cross-Attention kann er gezielt auf bestimmte Abschnitte im Audio zugreifen, die für das nächste Wort wichtig sind.

**4. Vorhersage des nächsten Tokens**

* Mit den Infos aus:
  + (a) bisherigem Text (Self-Attention)
  + (b) Encoder-Ausgabe (Cross-Attention)
* berechnet der Decoder, welches Token (Wort, Silbe oder Zeichen) am wahrscheinlichsten als Nächstes kommt.
* Beispiel: Nach SOT → „Guten“ → „Morgen“ → „, “ → „wie“ etc.

**5. Token anhängen und wiederholen**

* Das neu erzeugte Token wird an die bisherige Sequenz angehängt.
* Der Prozess wiederholt sich: Der Decoder nimmt jetzt SOT + „Guten“ als Input, sagt als Nächstes vielleicht „Morgen“ voraus, dann „, “ usw.

**Zusammenfassung:**  
Der Decoder liest das Audio (über den Encoder), baut daraus den gesprochenen Text Schritt für Schritt auf und nutzt dabei Wahrscheinlichkeit, Kontext und Audio-Informationen, um den bestmöglichen Text zu erzeugen.

**Training Data**

The models are trained on 680,000 hours of audio and the corresponding transcripts collected from the internet. 65% of this data (or 438,000 hours) represents English-language audio and matched English transcripts, roughly 18% (or 126,000 hours) represents non-English audio and English transcripts, while the final 17% (or 117,000 hours) represents non-English audio and the corresponding transcript. This non-English data represents 98 different languages.