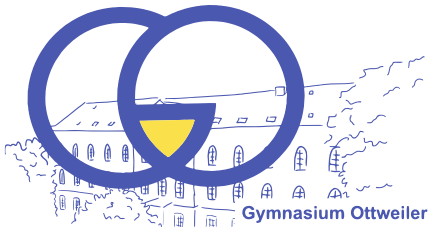
Kapitel 4

Sprachmodelle

Skript Informatik · Klasse 10 · MINT



Chatbots schreiben Aufsätze, beantworten Fragen und übersetzen Texte. Doch wie kann ein Computer Sätze bilden, die so klingen, als hätte ein Mensch sie geschrieben? In diesem Kapitel findest du es heraus – an einem einfachen Modell, das du vollständig verstehen und sogar von Hand nachrechnen kannst: dem **n-Gramm-Modell**. Damit erklärst du auch, warum solche Systeme manchmal überzeugend klingende, aber falsche Dinge schreiben.

Das erwartet dich in diesem Kapitel:

1. **Regeln oder Statistik?** – vom regelbasierten Chatbot ELIZA zu statistischen Sprachmodellen.
2. **n-Gramme zählen** – ein kleines Sprachmodell von Hand trainieren.
3. **Textanalyse mit dem Computer** – größere Texte mit Python auszählen.
4. **Texte erzeugen** – Wort für Wort mit next word prediction und verschiedenen Sampling-Strategien.
5. **Halluzinationen, moderne KI und die Arbeitswelt** – warum Sprachmodelle Falsches erfinden, was moderne Chatbots anders machen und wie KI Berufe verändert.

So kannst du es sagen

Mit diesen Satzanfängen beschreibst und begründest du deine Ergebnisse in diesem Kapitel:

- *„Auf den Kontext … folgt mit der Häufigkeit/Wahrscheinlichkeit … das Wort …“*
- *„Der erzeugte Text verändert sich, weil …“*
- *„Im Vergleich zum n-Gramm-Modell kann das LLM …, jedoch …“*

Am Ende hilft dir ein **Rückblick** mit den wichtigsten Begriffen und einem Kompetenz-Check beim Wiederholen.

4.1 Sprachmodelle: Regeln oder Statistik?

Leitidee

Du schreibst einem Chatbot eine Frage, und Sekunden später steht eine Antwort da, die klingt, als hätte ein Mensch sie geschrieben. Aber woher „weiß“ das Programm, was es antworten soll? Es gibt dafür zwei grundverschiedene Ideen. Die erste: Menschen schreiben **Regeln** auf, nach denen das Programm reagiert. Die zweite: Das Programm **zählt** in sehr vielen Texten, welche Wörter oft aufeinanderfolgen, und rechnet damit. In diesem Abschnitt lernst du beide Ideen kennen. Du programmierst einen kleinen Gesprächspartner nach festen Regeln und findest heraus, wo seine Grenzen liegen.

Lernziele

Nach diesem Abschnitt kannst du

- erläutern, was ein Sprachmodell ist,
- beschreiben, wie ein regelbasiertes System wie ELIZA funktioniert,
- ein kleines regelbasiertes Programm in Python testen und um eigene Regeln erweitern,
- die Grundidee eines n-Gramm-Modells beschreiben,
- regelbasierte Systeme und n-Gramm-Modelle unterscheiden und ihre Grenzen benennen.

4.1.1 Was ist ein Sprachmodell?

Menschen lernen Sprache, indem sie zuhören, lesen und sprechen. Dabei bekommst du mit der Zeit ein Gefühl dafür, welche Wörter zusammenpassen. Lies den folgenden Satzanfang:

> Ich trinke morgens eine Tasse …

Wahrscheinlich hast du sofort „Kaffee“ oder „Tee“ ergänzt – und nicht „Fahrrad“. Niemand hat dir dafür eine Regel beigebracht. Du hast diesen Satzanfang einfach schon sehr oft mit genau diesen Wörtern gehört oder gelesen.

Ein Computer kann so etwas nicht „fühlen“. Man kann ihm aber ein **Modell** der Sprache mitgeben: eine vereinfachte Beschreibung, wie Sätze aufgebaut sind und welche Wörter wie oft zusammen vorkommen. Damit kann ein Programm Texte untersuchen oder selbst neue Texte erzeugen. So ein Programm heißt **Sprachmodell**.

Definition: Sprachmodell

Ein **Sprachmodell** ist ein Computersystem, das ein vereinfachtes Modell einer Sprache enthält: wie Sätze aufgebaut sind und wie wahrscheinlich bestimmte Wortfolgen sind. Damit kann es Texte untersuchen oder neue Texte erzeugen.

Das Wort *Wahrscheinlichkeit* kennst du aus Mathe: Sie gibt an, wie sicher ein Ereignis eintritt, als Zahl zwischen 0 und 1 oder in Prozent. Bei Sprachmodellen geht es zum Beispiel um die Frage: Wie wahrscheinlich ist es, dass nach „eine Tasse“ das Wort „Tee“ folgt?

Sprachmodelle gibt es schon seit Jahrzehnten. Historisch gab es zwei wichtige Ansätze, die du in diesem Abschnitt kennenlernst: **regelbasierte Systeme** und **n-Gramm-Modelle**. Die heutigen großen Chatbots bauen auf dem zweiten Ansatz auf.

Merke

Ein **Sprachmodell** ist ein System, das sprachliche Strukturen bzw. Wahrscheinlichkeiten erfasst, um Texte zu analysieren oder neu zu erzeugen.

4.1.2 Regelbasierte Systeme: ELIZA

Ein Programm, das zuhört – scheinbar

Im Jahr 1966 veröffentlichte der Informatiker **Joseph Weizenbaum** am Massachusetts Institute of Technology (MIT) in den USA ein Programm namens **ELIZA**. Man konnte ihm Sätze eintippen, und es antwortete. In seiner bekanntesten Fassung spielte ELIZA eine Psychotherapeutin, die vor allem Rückfragen stellt. Ein Gespräch konnte etwa so aussehen:

```
Mensch:  Ich fühle mich müde.
ELIZA:   Warum fühlst du dich müde?
Mensch:  Ich brauche mehr Schlaf.
ELIZA:   Warum brauchst du mehr Schlaf?
```

Das wirkt, als würde ELIZA zuhören und mitdenken. Tatsächlich versteht das Programm **kein einziges Wort**. Es sucht im eingegebenen Satz nach bestimmten **Mustern**, zum Beispiel nach „Ich fühle mich“. Findet es ein Muster, setzt es den Rest des Satzes in eine vorbereitete Antwort ein. Weizenbaum war erschrocken, wie ernst manche Menschen das Programm trotzdem nahmen und ihm persönliche Dinge anvertrauten.

Das Wissen von ELIZA steckt also vollständig in Regeln, die ein Mensch vorher aufgeschrieben hat. Jede Regel hat die Form **„Wenn ..., dann ...“** – genau wie eine Verzweigung mit `if` in Python.

Definition: Regelbasiertes System

Ein **regelbasiertes System** reagiert nach **fest programmierten Regeln**. Es erkennt Muster in der Eingabe (z. B. bestimmte Wörter) und gibt die Antwort aus, die für dieses Muster vorgesehen ist.

Merke

Regelbasierte Systeme (z. B. das historische Programm **ELIZA**, 1966) reagieren auf Basis **fest programmierter Regeln** und Mustererkennung, z. B.: „Wenn der Satz ‚Ich fühle mich ...‘ enthält, antworte mit ‚Warum fühlst du dich ...?’ “. Sie „verstehen“ die Sprache nicht, sondern wenden feste Muster an.

Ein Werkzeug: Textteile ersetzen

Für unser eigenes kleines ELIZA brauchst du ein neues Werkzeug. Aus dem Satz „ich fühle mich müde“ soll das Programm den Rest „müde“ herausholen. Das geht, indem man den Anfang „ich fühle mich “ durch **nichts** ersetzt.

Werkzeug

Mit `text.replace(alt, neu)` erhältst du eine Kopie von `text`, in der jedes Vorkommen von `alt` durch `neu` ersetzt ist. Der ursprüngliche Text bleibt unverändert. Ersetzt du durch die leere Zeichenkette `""`, wird der Teil einfach entfernt:

```python
satz = "ich fühle mich müde"
print(satz.replace("fühle", "bin"))
print(satz.replace("ich fühle mich ", ""))
```

AUSGABE

```
ich bin mich müde
müde
```

Achte auf das Leerzeichen am Ende von `"ich fühle mich "`. Ohne dieses Leerzeichen würde der Rest mit einem Leerzeichen beginnen: `" müde"`.

Ein Mini-ELIZA in Python

Das folgende Programm ist ein stark vereinfachtes ELIZA. Es kennt nur vier Regeln:

Wenn der Satz ... enthält	antwortet Mini-ELIZA
„ich fühle mich …“	„Warum fühlst du dich …?“
„ich brauche …“	„Warum brauchst du …?“
ein Fragezeichen	„Interessante Frage - was denkst du selbst darüber?“
sonst	„Erzähl mir mehr davon.“

Die Regeln werden von oben nach unten geprüft. Die erste passende Regel gewinnt. Im Programm wird daraus eine Funktion mit `if`, `elif` und `else`:

```python
def mini_eliza(satz):
    satz = satz.lower()
    if "ich fühle mich" in satz:
        gefuehl = satz.replace("ich fühle mich ", "")
        return f"Warum fühlst du dich {gefuehl}?"
    elif "ich brauche" in satz:
        bedarf = satz.replace("ich brauche ", "")
        return f"Warum brauchst du {bedarf}?"
    elif "?" in satz:
        return "Interessante Frage - was denkst du selbst darüber?"
    else:
        return "Erzähl mir mehr davon."

print(mini_eliza("Ich fühle mich müde"))
print(mini_eliza("Ich brauche mehr Zeit"))
print(mini_eliza("Wie geht es dir?"))
print(mini_eliza("Die Sonne scheint"))
```

AUSGABE

```
Warum fühlst du dich müde?
Warum brauchst du mehr zeit?
Interessante Frage - was denkst du selbst darüber?
Erzähl mir mehr davon.
```

- **Zeile 1:** Die Funktion `mini_eliza` bekommt einen Satz als Parameter und gibt eine Antwort mit `return` zurück.
- **Zeile 2:** Der Satz wird in Kleinbuchstaben umgewandelt. So erkennt das Programm „Ich fühle mich“ und „ich fühle mich“ gleichermaßen.

- **Zeilen 3–5:** Die erste Regel. Steht „ich fühle mich“ im Satz, wird dieser Teil entfernt. Was übrig bleibt (hier „müde“), wird mit einem f-String in die Antwort eingesetzt.
- **Zeilen 6–8:** Die zweite Regel funktioniert genauso mit „ich brauche“.
- **Zeilen 9–10:** Enthält der Satz ein Fragezeichen, kommt eine allgemeine Antwort, die zum Nachdenken auffordert.
- **Zeilen 11–12:** Passt keine Regel, antwortet das Programm mit einem Satz, der fast immer irgendwie passt.
- **Zeilen 14–17:** Vier Testsätze. Für jeden wird die Antwort ausgegeben.

Schau dir die zweite Ausgabezeile genau an: „mehr zeit“ ist kleingeschrieben. Das Programm hat den ganzen Satz in Zeile 2 kleingeschrieben und setzt den Rest unverändert ein. Ein Mensch würde diesen Fehler nie machen. Das Programm merkt ihn nicht, denn es weiß nicht, dass „Zeit“ ein Nomen ist.

Versuch: Probiere es aus

Tippe das Programm ab und teste es mit eigenen Sätzen. Ändere auch die Reihenfolge der Regeln: Was antwortet das Programm auf „Ich fühle mich gut, und du?“, wenn die Regel mit dem Fragezeichen ganz oben steht?

Stolperstein

Wenn du `mini_eliza` mit `print` aufrufst und statt einer Antwort `None` erscheint, fehlt in einem Zweig das `return`. Dann gibt die Funktion nichts zurück. Prüfe, ob in **jedem** Zweig (`if`, `elif`, `else`) eine `return`-Anweisung steht.

Wo die Regeln an ihre Grenzen stoßen

Mini-ELIZA wirkt nur dann klug, wenn der Satz genau zu einem Muster passt. Schon kleine Änderungen bringen es aus dem Tritt. Die Funktion `mini_eliza` von oben wird hier mit zwei neuen Sätzen aufgerufen:

```
print(mini_eliza("Heute fühle ich mich müde"))
print(mini_eliza("Ich brauche dich"))
```

AUSGABE

```
Erzähl mir mehr davon.
Warum brauchst du dich?
```

Im ersten Satz stehen die Wörter in anderer Reihenfolge („fühle ich mich“). Das Muster passt nicht, und das Programm weicht auf seine Allerweltsantwort aus. Im zweiten Satz passt das Muster, aber das Programm übernimmt „dich“ wörtlich. Richtig wäre „Warum brauchst du mich?“. Um das zu können, müsste das Programm wissen, wer mit „dich“ gemeint ist. Das weiß es nicht.

Natürlich könnte man für jeden dieser Fälle eine weitere Regel schreiben. Aber Sprache ist unglaublich vielfältig: Es gibt unzählige Arten, dasselbe zu sagen. Man müsste unüberschaubar viele Regeln schreiben – und die Regeln dürfen sich nicht widersprechen.

Arbeitsauftrag 1: Grenzen des regelbasierten Ansatzes

Teste `mini_eliza` mit mindestens **drei eigenen Sätzen**, die **nicht** zu den vorgesehenen Mustern passen oder bei denen eine seltsame Antwort entsteht. Notiere Satz und Antwort. Beschreibe jeweils, warum das System hier keine sinnvolle Antwort liefern kann.

Meine Testsätze, Antworten und Erklärungen:

Arbeitsauftrag 2: Regel erweitern

Erweitere `mini_eliza` um mindestens **eine weitere Regel** deiner Wahl, zum Beispiel eine Reaktion auf „Ich mag …“ oder „Ich hasse …“. Nenne die neue Funktion `mini_eliza_erweitert` und teste sie mit passenden Sätzen.

4.1.3 n-Gramm-Modelle: Statistik statt Regeln

Die Idee: Zählen, was oft zusammen vorkommt

Erinnere dich an den Satzanfang „Ich trinke morgens eine Tasse …“. Du hast das fehlende Wort nicht mit einer Regel gefunden, sondern aus **Erfahrung**: Nach „eine Tasse“ kam in deinem Leben oft „Kaffee“ oder „Tee“.

Genau diese Idee nutzt ein **n-Gramm-Modell**. Statt Regeln aufzuschreiben, gibt man dem Computer sehr viele Texte. Er **zählt**, welche Wörter wie oft direkt hintereinander stehen. Eine solche Folge von aufeinanderfolgenden Wörtern heißt **n-Gramm**. Das *n* steht für die Anzahl der Wörter: Ein Wortpaar wie „eine Tasse“ ist ein 2-Gramm, man sagt auch **Bigramm**.

Ein ganz kleines Beispiel. Der Computer bekommt diese drei Sätze als Trainingstext:

```
ich trinke eine tasse tee
ich trinke eine tasse kaffee
ich trinke eine tasse tee
```

Nach dem Wort „tasse“ folgt zweimal „tee“ und einmal „kaffee“. Soll das Modell den Satz „ich trinke eine tasse …“ fortsetzen, ist „tee“ das wahrscheinlichste nächste Wort. Das Modell hat keine Regel über Getränke gelernt. Es hat nur gezählt.

Diese Vorhersage des nächsten Wortes heißt auf Englisch **next word prediction**. Die Wortvorschläge, die dir beim Tippen auf dem Handy über der Tastatur angezeigt werden, beruhen auf einer ähnlichen Idee: Das Programm schlägt Wörter vor, die nach dem bisher Getippten wahrscheinlich sind.

Definition: n-Gramm-Modell

Ein **n-Gramm-Modell** ist ein Sprachmodell, das aus einer großen Textsammlung zählt, wie häufig bestimmte Folgen von n Wörtern (n-Gramme) vorkommen. Aus diesen Häufigkeiten berechnet es, welches Wort wahrscheinlich als nächstes folgt.

Merke

n-Gramm-Modelle basieren auf der **statistischen Analyse großer Textmengen**: Sie erfassen, wie häufig bestimmte Wortfolgen (n-Gramme) in einer Textsammlung vorkommen, und nutzen diese Häufigkeiten zur Vorhersage des nächsten Wortes.

Wie man n-Gramme genau zählt und daraus Wahrscheinlichkeiten berechnet, lernst du im nächsten Abschnitt – zuerst von Hand, dann mit Python.

Und die modernen Chatbots?

Die Chatbots, die du vielleicht kennst, beruhen auf sogenannten **Large Language Models**, kurz **LLMs** (englisch für „große Sprachmodelle“). Sie wurden mit riesigen Textmengen trainiert und benutzen sehr komplexe Rechenverfahren, sogenannte **neuronale Netze**. Wie diese im Einzelnen funktionieren, ist nicht Thema dieses Skripts. Die Grundidee ist aber dieselbe wie beim n-Gramm-Modell: Ein LLM schätzt, welches Wort (genauer: welcher Wortbaustein) als nächstes am wahrscheinlichsten passt, hängt es an und wiederholt das immer wieder.

Gymnasium Ottweiler

Merke

Moderne **Large Language Models (LLMs)** nutzen komplexe neuronale Netze, beruhen aber im Kern auf einem ähnlichen Prinzip: Sie schätzen **Wahrscheinlichkeiten für die Fortsetzung eines Textes**.

Deshalb lohnt es sich, das n-Gramm-Modell genau zu verstehen. An ihm kannst du vollständig nachrechnen, was im Inneren passiert – bei einem LLM ist das nicht mehr möglich.

4.1.4 Regeln und Statistik im Vergleich

Die beiden Ansätze unterscheiden sich vor allem darin, **woher das Wissen** des Systems kommt:

	regelbasiertes System	**n-Gramm-Modell**
Woher kommt das Wissen?	Ein Mensch schreibt Regeln auf.	Der Computer zählt in Texten.
Was steht im Programm?	„Wenn …, dann …“-Regeln	Tabellen mit Häufigkeiten
Neue Formulierungen	nur, wenn eine Regel passt	möglich, wenn ähnliche Wortfolgen im Text vorkamen
Verhalten nachvollziehbar?	ja, jede Antwort folgt aus einer Regel	ja, jede Wahl folgt aus den Zahlen
Versteht es die Bedeutung?	nein	nein

Die letzte Zeile ist wichtig: **Keines** der beiden Systeme versteht, was es schreibt. Das regelbasierte System erkennt Zeichenfolgen, das n-Gramm-Modell rechnet mit Häufigkeiten.

So kannst du es sagen

So beschreibst du die Ansätze in Fachsprache – vom einfachen zum genauen Satz:

- *„ELIZA hat Wenn-dann-Sätze.“ → „ELIZA ist ein regelbasiertes System: Es erkennt Muster in der Eingabe und reagiert nach fest programmierten Regeln.“*
- *„Das Modell guckt, was oft kommt.“ → „Das n-Gramm-Modell ermittelt aus einer Textsammlung, wie häufig Wortfolgen vorkommen, und sagt daraus das wahrscheinlichste nächste Wort voraus.“*
- *„Der Chatbot versteht nichts.“ → „Das System verarbeitet Zeichenfolgen bzw. Häufigkeiten, ohne die Bedeutung der Wörter zu erfassen.“*

4.1.5 Übungen

Arbeitsauftrag 3: Regeln oder Statistik?

Entscheide jeweils, ob das beschriebene Programm **regelbasiert** arbeitet oder nach dem Prinzip eines **n-Gramm-Modells**. Begründe in einem Satz.

1. Ein Programm antwortet auf jede E-Mail, die das Wort „Urlaub“ enthält, mit „Ich bin bis Montag nicht erreichbar.“
2. Ein Programm schlägt beim Tippen das Wort vor, das in einer großen Sammlung von Nachrichten am häufigsten nach dem zuletzt getippten Wort stand.
3. Ein Spiel-Chatbot erkennt die Wörter „hallo“, „tschüss“ und „hilfe“ und hat für jedes eine feste Antwort.

4. Ein Programm zählt in allen Märchen der Brüder Grimm, welches Wort wie oft nach „es war“ folgt, und ergänzt damit angefangene Sätze.

Meine Antworten:

Arbeitsauftrag 4: Vergleich in Worten

Beantworte schriftlich:

1. Warum stößt ein rein regelbasiertes System bei der Vielfalt natürlicher Sprache schnell an Grenzen?
2. Welchen grundsätzlich anderen Ansatz verfolgen n-Gramm-Modelle (Stichwort: Regeln oder Statistik)?

Diese Fragen werden in den nächsten Abschnitten praktisch vertieft.

Gymnasium Ottweiler

Meine Antwort:

4.1.6 Weitere Übungen

Hier kannst du das Gelernte weiter üben. Die Punkte im Titel zeigen, wie schwer eine Aufgabe ist: ●○○ leicht, ●●○ mittel, ●●● knifflig. Fang mit den leichten an.

Arbeitsauftrag 5: Richtig oder falsch? ●○○

Entscheide, ob die Aussage richtig oder falsch ist. Trage „r“ oder „f“ ein. Verbessere die falschen Aussagen unten in einem Satz.

Nr.	Aussage	r oder f?
1	Ein Sprachmodell erfasst sprachliche Strukturen bzw. Wahrscheinlichkeiten, um Texte zu analysieren oder zu erzeugen.	
2	ELIZA versteht die Bedeutung der Sätze, die man eintippt.	
3	Ein regelbasiertes System reagiert nach Regeln, die vorher fest programmiert wurden.	
4	Ein n-Gramm-Modell zählt in Texten, wie oft bestimmte Wortfolgen vorkommen.	
5	Ein n-Gramm-Modell braucht für jede mögliche Eingabe eine eigene Wenn-dann-Regel.	
6	Moderne LLMs arbeiten ganz anders als n-Gramm-Modelle: Sie schätzen keine Wahrscheinlichkeiten.	

Verbesserungen:

Gymnasium Ottweiler

Arbeitsauftrag 6: Was antwortet Mini-ELIZA? ●○○

Bestimme **ohne Computer**, was die Funktion `mini_eliza` aus dem Abschnitt *Ein Mini-ELIZA in Python* auf die folgenden Sätze antwortet. Denke daran: Der Satz wird zuerst kleingeschrieben, und die Regeln werden von oben nach unten geprüft. Die erste passende Regel gewinnt.

Satz	Antwort von `mini_eliza`
Ich fühle mich super	
Ich brauche eine Pause	
Hast du Hunger?	
Mein Handy ist kaputt	
Ich brauche Hilfe, kannst du mir helfen?	

Arbeitsauftrag 7: Regeln oder Statistik? ●○○

Entscheide jeweils, ob das Programm **regelbasiert** arbeitet oder nach dem Prinzip eines **n-Gramm-Modells**. Begründe kurz.

1. Ein Chatbot eines Online-Shops schickt auf jede Nachricht, in der „Rücksendung“ vorkommt, einen Link zum Rücksendeformular.
2. Ein Programm hat Tausende Chatnachrichten ausgewertet. Nach „bis“ schlägt es das Wort vor, das dort am häufigsten folgte, zum Beispiel „morgen“.
3. Ein Lernspiel prüft, ob deine Antwort das Wort „Photosynthese“ enthält. Wenn ja, zeigt es „Richtig!“.
4. Ein Programm hat alle Bände einer Fantasy-Buchreihe ausgezählt. Es setzt angefangene Sätze fort, indem es immer ein Wort wählt, das dort häufig auf die letzten Wörter folgte.
5. Ein Sprachassistent im Auto zeigt die nächste Tankstelle an, sobald im gesprochenen Satz das Wort „Tankstelle“ vorkommt.

Gymnasium Ottweiler

Meine Antworten:

Arbeitsauftrag 8: Der Kiosk-Bot ●●○

Der Schulkiosk hat einen kleinen regelbasierten Chatbot:

```python
def kiosk_bot(satz):
    satz = satz.lower()
    if "brötchen" in satz:
        return "Brötchen gibt es ab 9:30 Uhr."
    elif "kostet" in satz:
        return "Alle Getränke kosten 1 Euro."
    elif "?" in satz:
        return "Das weiß ich leider nicht."
    else:
        return "Hallo! Frag mich etwas zum Kiosk."
```

1. Bestimme **ohne Computer** die Antworten auf die Sätze in der Tabelle. Trage ein, ob die Antwort zur Frage passt (ja/nein).
2. Erkläre bei jeder unpassenden Antwort, warum das Programm sie gibt.

Satz	Antwort	passt?
Wann gibt es Brötchen?		
Was kostet ein Brötchen?		
Wie teuer ist ein Kakao?		
Was kostet ein Wasser?		
Ich hätte gern einen Kakao.		

Meine Erklärungen:

Arbeitsauftrag 9: Regeln für die Schulbibliothek ●●○

Die Schulbibliothek möchte einen regelbasierten Chatbot.

1. Formuliere **drei Regeln** in der Form „Wenn der Satz … enthält, antworte: …“.
2. Nenne **zwei Sätze**, die eine Schülerin schreiben könnte, bei denen deine Regeln eine unpassende oder gar keine passende Antwort liefern. Erkläre jeweils, warum.

Meine Regeln und Beispiele:

Gymnasium Ottweiler

Arbeitsauftrag 10: Den Kiosk-Bot verbessern ●●●

Schreibe am Computer eine Funktion `kiosk_bot_2`, die zwei Schwächen des Kiosk-Bots behebt:

1. Fragt jemand nach dem **Preis eines Brötchens** (der Satz enthält „brötchen“ und „kostet“), antwortet sie: „Ein Brötchen kostet 50 Cent.“
2. Das Wort „teuer“ soll genauso behandelt werden wie „kostet“.

Alle anderen Antworten bleiben wie beim Kiosk-Bot. Teste deine Funktion mit den fünf Sätzen aus dem Auftrag *Der Kiosk-Bot* und zusätzlich mit „Ist ein Brötchen teuer?“.

Tipp

Überlege genau, in welcher **Reihenfolge** die Regeln stehen müssen. Eine Regel, die zwei Bedingungen braucht, muss vor den Regeln stehen, die nur eine davon prüfen. Bedingungen verknüpfst du mit `and` und `or`. Klammern helfen, den Überblick zu behalten.

4.1.7 Das Wichtigste in Kürze

- Ein **Sprachmodell** erfasst sprachliche Strukturen bzw. Wahrscheinlichkeiten, um Texte zu analysieren oder zu erzeugen.
- **Regelbasierte Systeme** wie ELIZA (1966) reagieren nach fest programmierten Wenn-dann-Regeln und erkennen nur Muster in der Eingabe.
- Regelbasierte Systeme scheitern an der Vielfalt der Sprache: Für jede Formulierung bräuchte man eine eigene Regel.
- **n-Gramm-Modelle** zählen in großen Textsammlungen, wie oft Wortfolgen (n-Gramme) vorkommen, und sagen damit das nächste Wort voraus (**next word prediction**).
- Moderne **LLMs** nutzen neuronale Netze, schätzen aber ebenfalls Wahrscheinlichkeiten für die Fortsetzung eines Textes.
- Keines dieser Systeme versteht die Bedeutung dessen, was es verarbeitet.

Selbstcheck

- ☐ Ich kann erläutern, was ein Sprachmodell ist.
- ☐ Ich kann beschreiben, wie ELIZA auf einen Satz reagiert.
- ☐ Ich kann eine Funktion wie `mini_eliza` testen und um eine Regel erweitern.
- ☐ Ich kann an Beispielen zeigen, wo regelbasierte Systeme an ihre Grenzen stoßen.
- ☐ Ich kann die Grundidee eines n-Gramm-Modells mit eigenen Worten erklären.
- ☐ Ich kann regelbasierte Systeme und n-Gramm-Modelle an einem Beispiel unterscheiden.

4.2 n-Gramme zählen: ein Sprachmodell von Hand trainieren

Leitidee

Ein n-Gramm-Modell lernt Sprache, indem es zählt. Das klingt einfach – und das ist es auch. In diesem Abschnitt machst du genau das, was der Computer beim **Training** eines n-Gramm-Modells tut: Du nimmst einen kurzen Text, zerlegst ihn in Wortpaare, zählst sie und berechnest daraus, mit welcher Wahrscheinlichkeit auf ein Wort ein bestimmtes anderes Wort folgt. Am Ende hast du ein kleines Sprachmodell auf Papier. Zur Kontrolle lässt du Python nachzählen.

Lernziele

Nach diesem Abschnitt kannst du

- erklären, was ein n-Gramm ist, und Unigramme, Bigramme und Trigramme unterscheiden,
- die Begriffe Textkorpus und Textkörperanalyse erläutern,
- in einem kurzen Text alle Bigramme von Hand ermitteln und zählen,
- aus den Häufigkeiten die Wahrscheinlichkeiten für Folgewörter berechnen und in einer Tabelle festhalten,
- eine Handauszählung mit einem Python-Programm überprüfen.

4.2.1 Was ist ein n-Gramm?

Nimm den Satz „der hund bellt laut“. Du kannst ihn auf verschiedene Arten in Stücke schneiden:

- in **einzelne Wörter**: „der“, „hund“, „bellt“, „laut“,
- in **Wortpaare**: „der hund“, „hund bellt“, „bellt laut“,
- in **Dreiergruppen**: „der hund bellt“, „hund bellt laut“.

Jedes dieser Stücke ist ein **n-Gramm**. Das *n* gibt an, wie viele Wörter direkt hintereinander zu einem Stück gehören. Die Stücke überlappen sich: Jedes neue Stück beginnt **ein Wort weiter rechts** als das vorige.

Name	n	Stücke aus „der hund bellt laut“	Anzahl
Unigramm	1	der · hund · bellt · laut	4
Bigramm	2	der hund · hund bellt · bellt laut	3
Trigramm	3	der hund bellt · hund bellt laut	2

Mit jedem Wort mehr pro Stück gibt es ein Stück weniger. Allgemein gilt: Ein Text mit N Wörtern hat $N-n+1$ n-Gramme. Für Bigramme ($n = 2$) sind das $N - 1$.

Merke

- Ein **n-Gramm** ist eine Folge von n aufeinanderfolgenden Wörtern (oder Zeichen) aus einem Text.
  - **Unigramm** ($n = 1$): einzelne Wörter
  - **Bigramm** ($n = 2$): Wortpaare
  - **Trigramm** ($n = 3$): Wortdreiergruppen

Statt Wörtern kann man auch **Zeichen** zu n-Grammen zusammenfassen. Die Zeichen-Bigramme von „hund“ sind „hu“, „un“ und „nd“. In diesem Kapitel geht es aber immer um Wörter.

4.2.2 Textkorpus und Textkörperanalyse

Ein n-Gramm-Modell braucht Text, aus dem es lernen kann. Diese Textsammlung heißt **Textkorpus** oder kurz **Korpus** (lateinisch *corpus* = Körper; deshalb sagt man auch **Textkörper**). Bei echten Sprachmodellen besteht der Korpus aus riesigen Mengen an Büchern, Zeitungsartikeln und Internetseiten. Bei uns sind es zunächst nur ein oder zwei Sätze. Das Prinzip ist dasselbe.

Definition: Textkorpus

Ein **Textkorpus** (auch Textkörper) ist eine Sammlung von Texten, die als **Trainingstext** für ein Sprachmodell dient.

Definition: Textkörperanalyse

Bei der **Textkörperanalyse** wird ermittelt, wie oft jedes n-Gramm in einem Textkorpus vorkommt. Aus diesen Häufigkeiten werden anschließend Wahrscheinlichkeiten berechnet.

Merke

Bei der **Textkörperanalyse** zählt man, wie oft jedes n-Gramm in einem **Textkorpus** (Trainingstext) vorkommt.

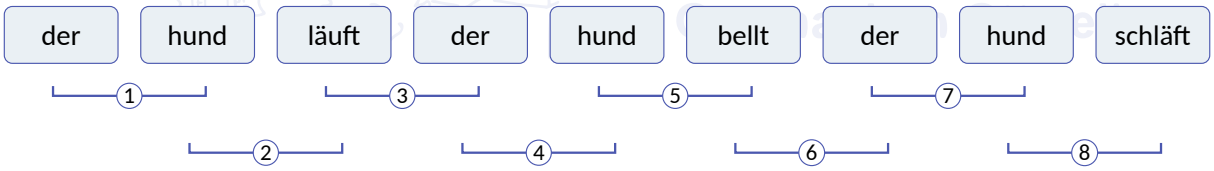
Damit das Zählen eindeutig ist, schreiben wir in diesem Kapitel alle Wörter **klein** und lassen Satzzeichen weg. Sonst wären „Der“ und „der“ für den Computer zwei verschiedene Wörter.

4.2.3 Bigramme von Hand zählen

Wir untersuchen diesen Textkorpus:

> der hund läuft der hund bellt der hund schläft

Der Text hat 9 Wörter, also gibt es $9 - 1 = 8$ Bigramme. Du findest sie, indem du ein „Fenster“ aus zwei Wörtern über den Text schiebst – immer um ein Wort weiter:



Ein Fenster aus zwei Wörtern wandert über den Text. Jede Klammer markiert ein Bigramm, die Zahl gibt die Reihenfolge an

Als Liste geschrieben sind das die acht Bigramme: der hund · hund läuft · läuft der · der hund · hund bellt · bellt der · der hund · hund schläft.

Jetzt wird gezählt. Am einfachsten geht das mit einer **Strichliste**: Du gehst die Liste von links nach rechts durch. Kommt ein Bigramm zum ersten Mal vor, bekommt es eine neue Zeile. Kommt es noch einmal vor, machst du einen weiteren Strich.

Bigramm	Strichliste	Anzahl
der hund	III	3
hund läuft	I	1
läuft der	I	1
hund bellt	I	1
bellt der	I	1
hund schläft	I	1
Summe		**8**

Die Summe muss genau die Anzahl der Bigramme ergeben, hier 8. Das ist eine gute Kontrolle, ob du keines vergessen hast.

Das Ergebnis: Auf das Wort „der“ folgt in diesem Text **dreimal** „hund“ – und nie etwas anderes. Das ist mit Abstand das häufigste Bigramm mit „der“ als erstem Wort. Auf „hund“ folgen dagegen drei verschiedene Wörter, jedes genau einmal.

Tipp

Beim Zählen von Hand passieren leicht Fehler. Nummeriere die Wörter des Textes vorher durch und streiche jedes Bigramm in der Liste ab, sobald du es gezählt hast.

Arbeitsauftrag 11: Von Hand zählen

Gegeben ist der Textkorpus:

> die katze schläft die katze spielt die katze frisst die katze schläft

1. Ermittle **von Hand** alle Bigramme und zähle, wie oft jedes vorkommt. Lege dazu eine Tabelle mit Strichliste an.
2. Welches Wort folgt im Korpus am häufigsten direkt auf „die“?
3. Welches Wort folgt am häufigsten auf „katze“?

Meine Auszählung:

4.2.4 Von der Häufigkeit zur Wahrscheinlichkeit

Häufigkeiten allein reichen noch nicht. Ein Sprachmodell will wissen: **Wenn** gerade das Wort „der“ dasteht – **wie wahrscheinlich** ist es dann, dass als nächstes „hund“ kommt? Dafür vergleicht man, wie oft „der hund“ vorkommt, mit der Anzahl **aller** Bigramme, die mit „der“ beginnen.

Wir rechnen das an einem neuen Korpus Schritt für Schritt durch. Er ist etwas abwechslungsreicher, damit interessantere Zahlen entstehen:

> der hund bellt der hund läuft der ball rollt der hund bellt

Beispiel: Wahrscheinlichkeiten Schritt für Schritt

Schritt 1: Wörter und Bigramme ermitteln. Der Text hat 12 Wörter, also $12 - 1 = 11$ Bigramme:

der hund · hund bellt · bellt der · der hund · hund läuft · läuft der · der ball · ball rollt · rollt der · der hund · hund bellt

Schritt 2: Bigramme zählen und nach dem ersten Wort ordnen. Alle Bigramme mit demselben ersten Wort stehen untereinander. Das erste Wort nennen wir **Vorgängerwort**, das zweite **Folgewort**.

Vorgängerwort	Folgewort	Anzahl
der	hund	3
der	ball	1
hund	bellt	2
hund	läuft	1
bellt	der	1
läuft	der	1
ball	rollt	1
rollt	der	1

Kontrolle: $3 + 1 + 2 + 1 + 1 + 1 + 1 + 1 = 11$. ✓

Schritt 3: Wie oft steht jedes Vorgängerwort vorne in einem Bigramm? Dazu addierst du die Anzahlen in seinen Zeilen: „der“ $3 + 1 = 4$, „hund“ $2 + 1 = 3$, „bellt“ 1, „läuft“ 1, „ball“ 1, „rollt“ 1.

Schritt 4: Wahrscheinlichkeit berechnen. Teile die Anzahl des Bigramms durch die Zahl aus Schritt 3. Für „der“, gefolgt von „hund“:

$$P(\text{der} \to \text{hund}) = \frac{3}{4} = 0{,}75 = 75\,\%$$

Für „hund“, gefolgt von „bellt“:

$$P(\text{hund} \to \text{bellt}) = \frac{2}{3} \approx 0{,}67 = 67\,\%$$

Das Zeichen P steht für *probability*, englisch für Wahrscheinlichkeit. $P(\text{der} \to \text{hund})$ liest du: „die Wahrscheinlichkeit, dass auf ‚der‘ das Wort ‚hund‘ folgt“.

Rechnest du das für alle Zeilen aus, erhältst du die vollständige **Wahrscheinlichkeitstabelle** des Modells:

Vorgängerwort	Folgewort	Anzahl	Rechnung	Wahrscheinlichkeit
der	hund	3	$\frac{3}{4}$	0,75 = 75 %
der	ball	1	$\frac{1}{4}$	0,25 = 25 %
hund	bellt	2	$\frac{2}{3}$	≈ 0,67 = 67 %
hund	läuft	1	$\frac{1}{3}$	≈ 0,33 = 33 %
bellt	der	1	$\frac{1}{1}$	1 = 100 %
läuft	der	1	$\frac{1}{1}$	1 = 100 %
ball	rollt	1	$\frac{1}{1}$	1 = 100 %
rollt	der	1	$\frac{1}{1}$	1 = 100 %

Für jedes Vorgängerwort ergeben die Wahrscheinlichkeiten zusammen genau 1, also 100 %: bei „der“ $0{,}75 + 0{,}25 = 1$, bei „hund“ $\frac{2}{3} + \frac{1}{3} = 1$. Das ist logisch, denn **irgendein** Folgewort kommt nach „der“ ja immer. Diese Summe ist deine Kontrolle.

Merke

Die **Wahrscheinlichkeit**, dass auf ein Wort A das Wort B folgt, berechnet man so:

$$P(A \to B) = \frac{\text{Anzahl des Bigramms „A B“}}{\text{Anzahl aller Bigramme, die mit A beginnen}}$$

Für jedes Vorgängerwort ergeben die Wahrscheinlichkeiten aller Folgewörter zusammen 1 (also 100 %).

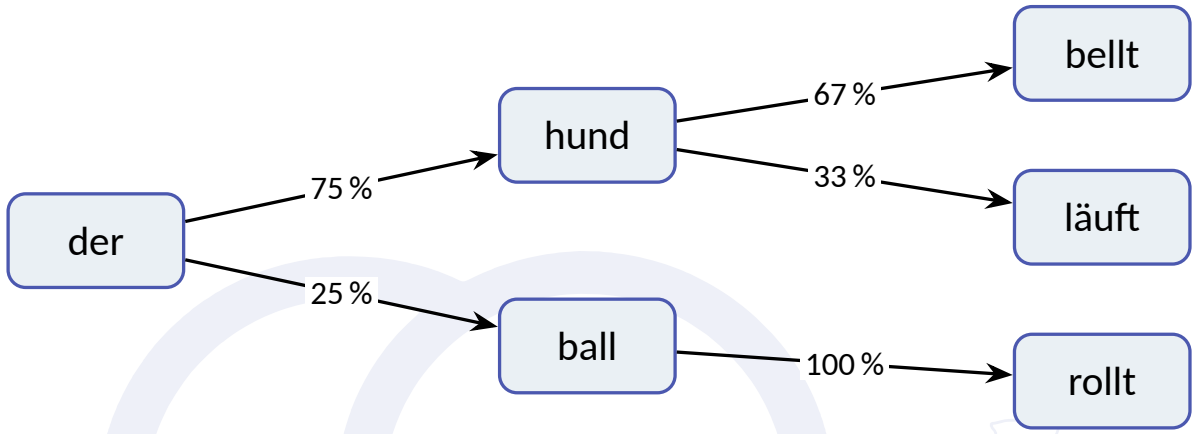
Gymnasium Ottweiler

Stolperstein

Im Nenner steht **nicht** die Gesamtzahl aller Bigramme im Text, sondern nur die Anzahl der Bigramme, die mit dem Vorgängerwort **beginnen**. $\frac{3}{11}$ wäre für „der hund“ also falsch.

Vorsicht auch beim letzten Wort: „bellt“ steht zweimal im Text. Das zweite „bellt“ ist aber das letzte Wort, danach folgt nichts mehr. Deshalb beginnt nur **ein** Bigramm mit „bellt“, und es gilt $P(\text{bellt} \to \text{der}) = \frac{1}{1}$, nicht $\frac{1}{2}$. Zähle im Nenner also nur Vorkommen, auf die noch ein Wort folgt.

Man kann ein solches Modell auch als **Wortgraph** zeichnen: Jedes Wort ist ein Kästchen, und ein Pfeil führt zu jedem möglichen Folgewort. Am Pfeil steht die Wahrscheinlichkeit.



Ausschnitt aus dem Bigramm-Modell zu „der hund bellt der hund läuft der ball rollt der hund bellt“: die Folgewörter von „der“, „hund“ und „ball“. Die Pfeile zurück zu „der“ sind weggelassen

Merke

Aus den Häufigkeiten lässt sich die **Wahrscheinlichkeit** ableiten, mit der auf ein bestimmtes Wort (bzw. eine Wortfolge) ein bestimmtes Folgewort folgt. Das ist die Grundlage für die spätere **Textgenerierung**.

So kannst du es sagen

So beschreibst du deine Ergebnisse in Fachsprache:

- *„Nach der kommt meistens hund.“* → *„Im Korpus folgt auf ‚der‘ in drei von vier Fällen ‚hund‘.“*
- *„hund bellt ist 67 %.“* → *„Die Wahrscheinlichkeit, dass auf ‚hund‘ das Wort ‚bellt‘ folgt, beträgt* $\frac{2}{3} \approx 67\,\%$*.“*

Arbeitsauftrag 12: Wahrscheinlichkeiten berechnen

Verwende deine Häufigkeiten aus dem Auftrag *Von Hand zählen* (Korpus „die katze schläft die katze spielt …“).

1. Berechne die **Wahrscheinlichkeit** für jedes Wort, das auf „die“ folgt: Anzahl des jeweiligen Bigramms geteilt durch die Gesamtanzahl aller Bigramme, die mit „die“ beginnen.

2. Berechne ebenso die Wahrscheinlichkeiten für alle Wörter, die auf „katze“ folgen. Gib sie als Bruch, als Dezimalzahl und in Prozent an.

Meine Rechnung:

Arbeitsauftrag 13: Fehler finden

Tim soll für den Korpus „ich mag tee ich mag kaffee ich trinke tee“ die Wahrscheinlichkeitstabelle aufstellen. Er schreibt:

Vorgängerwort	Folgewort	Wahrscheinlichkeit
ich	mag	$\frac{2}{8}$
ich	trinke	$\frac{1}{8}$
tee	ich	$\frac{1}{2}$

1. Beschreibe Tims zwei Fehler.
2. Stelle die vollständige und richtige Wahrscheinlichkeitstabelle für diesen Korpus auf.

Fehler und richtige Tabelle:

Gymnasium Ottweiler

4.2.5 Trigramme: mehr Wörter als Kontext

Bei Bigrammen hängt das Folgewort nur vom **einen** Wort davor ab. Bei Trigrammen schaut man auf die **zwei** Wörter davor. Diese Wörter, auf die das Modell beim Vorhersagen schaut, heißen **Kontext**.

Im Korpus „der hund bellt der hund läuft der ball rollt der hund bellt“ gibt es $12 - 2 = 10$ Trigramme. Drei davon beginnen mit dem Kontext „der hund“:

Kontext	Folgewort	Anzahl	Wahrscheinlichkeit
der hund	bellt	2	$\frac{2}{3} \approx 67\,\%$
der hund	läuft	1	$\frac{1}{3} \approx 33\,\%$

Gerechnet wird genau wie bei Bigrammen: Anzahl des Trigramms geteilt durch die Anzahl aller Trigramme mit demselben Kontext. In diesem kleinen Text ändert sich dadurch noch nichts. In langen Texten macht der längere Kontext aber einen großen Unterschied: Nach „die“ kann fast alles kommen, nach „die katze“ schon viel weniger. Wie sich das auf erzeugte Texte auswirkt, untersuchst du im Abschnitt über Textgenerierung.

4.2.6 Kontrolle mit Python

Bei kurzen Texten geht das Zählen von Hand. Bei langen Texten ist es mühsam und fehleranfällig. Hier hilft ein Programm. Zuerst muss der Text in einzelne Wörter zerlegt werden.

Werkzeug

Mit `text.split()` zerlegst du eine Zeichenkette an den Leerzeichen in ihre Wörter. Das Ergebnis ist eine **Liste** von Zeichenketten. Mehrere Leerzeichen und Zeilenumbrüche hintereinander zählen dabei wie ein einziges Leerzeichen.

```
satz = "der hund bellt laut"
woerter = satz.split()
print(woerter)
print(len(woerter))
```

AUSGABE

```
['der', 'hund', 'bellt', 'laut']
4
```

Mit dieser Liste kannst du alle Bigramme ausgeben. Das Bigramm Nummer `i` besteht aus dem Wort an der Stelle `i` und dem Wort an der Stelle `i + 1`:

```
korpus = "der hund läuft der hund bellt der hund schläft"
woerter = korpus.split()
for i in range(len(woerter) - 1):
    print(woerter[i], woerter[i + 1])
```

AUSGABE

```
der hund
hund läuft
läuft der
der hund
hund bellt
bellt der
der hund
hund schläft
```

- **Zeile 2:** `woerter` ist jetzt die Liste `['der', 'hund', 'läuft', …]` mit 9 Elementen. Die Indizes laufen von 0 bis 8.
- **Zeile 3:** Die Schleife läuft nur bis `len(woerter) - 1`, also für `i` von 0 bis 7. Das sind genau 8 Durchläufe – so viele, wie es Bigramme gibt.
- **Zeile 4:** Ausgegeben werden das Wort an Stelle `i` und sein Nachfolger an Stelle `i + 1`. Das ist das „Fenster“ aus der Abbildung oben.

Stolperstein

Schreibst du in der Schleife nur `range(len(woerter))`, läuft `i` bis zum letzten Index. Im letzten Durchlauf will das Programm dann das Wort **nach** dem letzten Wort lesen. Das gibt es nicht. Hier ein kurzes Beispiel mit drei Wörtern (Indizes 0 bis 2): Im dritten Durchlauf ist `i = 2`, und `woerter[3]` existiert nicht.

```
1 woerter = "der hund bellt".split()
2 for i in range(len(woerter)):
3     print(woerter[i], woerter[i + 1])
```

AUSGABE

```
der hund
hund bellt
```

FEHLERMELDUNG

```
IndexError: list index out of range
```

Das letzte Wort hat keinen Nachfolger. Deshalb gibt es ein Bigramm weniger als Wörter.

Jetzt soll das Programm auch zählen. Dazu speichern wir jedes Bigramm als Zeichenkette aus zwei Wörtern, zum Beispiel `"die katze"`, in einer Liste. Dann gehen wir so vor wie bei der Strichliste: Zuerst sammeln wir alle **verschiedenen** Bigramme, dann zählen wir für jedes, wie oft es in der Liste vorkommt.

```
1  def zaehle(liste, gesucht):
2      anzahl = 0
3      for element in liste:
4          if element == gesucht:
5              anzahl += 1
6      return anzahl
7
8  korpus = "die katze schläft die katze spielt "
9  korpus = korpus + "die katze frisst die katze schläft"
10 woerter = korpus.split()
11
12 bigramme = []
13 for i in range(len(woerter) - 1):
14     bigramme.append(woerter[i] + " " + woerter[i + 1])
15
16 verschiedene = []
17 for bigramm in bigramme:
18     if bigramm not in verschiedene:
19         verschiedene.append(bigramm)
20
21 for bigramm in verschiedene:
22     print(bigramm, ":", zaehle(bigramme, bigramm))
```

AUSGABE

```
die katze : 4
katze schläft : 2
schläft die : 1
katze spielt : 1
spielt die : 1
katze frisst : 1
frisst die : 1
```

- **Zeilen 1–6:** Eine solche Zählfunktion kennst du von den Listenalgorithmen. `zaehle` zählt, wie oft `gesucht` in der Liste vorkommt.
- **Zeilen 8–10:** Der Korpus ist für eine Zeile zu lang. Deshalb wird er in zwei Teilen gebildet und mit `+` zusammengesetzt. Dann wird er in Wörter zerlegt.
- **Zeilen 12–14:** Für jede Position `i` wird das Bigramm als Zeichenkette gebildet, z. B. `"die" + " " + "katze"` ergibt `"die katze"`, und an die Liste `bigramme` angehängt. Sie enthält danach alle 11 Bigramme, auch die doppelten.
- **Zeilen 16–19:** In `verschiedene` kommt jedes Bigramm nur einmal, nämlich beim ersten Auftreten. Das entspricht dem Anlegen einer neuen Zeile in der Strichliste.
- **Zeilen 21–22:** Für jedes verschiedene Bigramm wird gezählt, wie oft es in `bigramme` steht.

Vergleiche die Ausgabe mit deiner Lösung zum Auftrag *Von Hand zählen*. Die Zahlen müssen übereinstimmen.

Versuch: Probiere es aus

Setze in den Zeilen 8 und 9 den Korpus „der hund bellt der hund läuft der ball rollt der hund bellt“ ein und vergleiche mit der Tabelle aus dem Beispiel *Wahrscheinlichkeiten Schritt für Schritt*. Ergänze dann eine Zeile, die die Anzahl aller Bigramme mit `len(bigramme)` ausgibt.

4.2.7 Übungen

Arbeitsauftrag 14 (Zusatz): Trigramme

Erweitere das Programm aus dem Abschnitt *Kontrolle mit Python* so, dass es **Trigramme** (Dreiergruppen) statt Bigramme zählt. Verwende deinen eigenen, etwas längeren Beispielsatz.

4.2.8 Weitere Übungen

Hier kannst du das Gelernte weiter üben. Die Punkte im Titel zeigen, wie schwer eine Aufgabe ist: ●○○ leicht, ●●○ mittel, ●●● knifflig. Fang mit den leichten an.

Arbeitsauftrag 15: n-Gramme aufschreiben ●○○

1. Bestimme für den Satz „wir fahren morgen nach saarbrücken“ alle Unigramme, Bigramme und Trigramme und gib jeweils die Anzahl an.
2. n-Gramme kann man auch aus **Zeichen** bilden. Bestimme die Zeichen-Bigramme des Wortes „kino“.

n-Gramme	Anzahl
Unigramme	
Bigramme	
Trigramme	
Zeichen-Bigramme von „kino“	

Arbeitsauftrag 16: Wie viele Bigramme? ●○○

Bestimme für jeden Text die Anzahl der Wörter, die Anzahl der Bigramme und die Anzahl der **verschiedenen** Bigramme.

Text	Wörter	Bigramme	verschiedene Bigramme
ich bin ich			
la la la la			
heute ist heute und morgen ist morgen			
du und ich und du und ich			

Arbeitsauftrag 17: Strichliste ●○○

Gegeben ist der Textkorpus:

> wir spielen fußball wir spielen basketball wir schauen fußball

1. Ermittle alle Bigramme und zähle sie mit einer Strichliste.
2. Welches Wort folgt am häufigsten auf „wir“?

Bigramm	Strichliste	Anzahl
Summe		

Arbeitsauftrag 18: Wahrscheinlichkeiten aus einer Tabelle ●○○

In einem großen Korpus wurde gezählt, welche Wörter auf „gehe“ folgen: 5-mal „ins“, 3-mal „nach“ und 2-mal „zum“. Andere Folgewörter gibt es nicht.

1. Berechne die Wahrscheinlichkeit für jedes Folgewort als Bruch, als Dezimalzahl und in Prozent.
2. Kontrolliere mit der Summe.
3. Welches Folgewort ist am wahrscheinlichsten?

Kontext	Folgewort	Anzahl	Bruch	Dezimalzahl	Prozent
gehe	ins	5			
gehe	nach	3			
gehe	zum	2			

Arbeitsauftrag 19: Ein Modell von Hand trainieren ●●○

Gegeben ist der Textkorpus:

> ich höre musik ich höre podcasts ich lese comics ich höre musik

1. Ermittle alle Bigramme und ihre Häufigkeiten.
2. Stelle die vollständige Wahrscheinlichkeitstabelle auf (Kontext, Folgewort, Anzahl, Wahrscheinlichkeit als Bruch und in Prozent).
3. Kontrolliere: Ergeben für jeden Kontext die Wahrscheinlichkeiten zusammen 1?

Meine Auszählung und meine Tabelle:

Arbeitsauftrag 20: Mehr Kontext mit Trigrammen ●●○

Gegeben ist der Textkorpus:

> er spielt gitarre sie spielt geige er spielt gitarre sie spielt klavier

1. Bestimme mit Bigrammen die Wahrscheinlichkeiten aller Folgewörter von „spielt“.
2. Bestimme alle Trigramme, die mit dem Kontext „er spielt“ beginnen, und alle, die mit dem Kontext „sie spielt“ beginnen. Berechne die Wahrscheinlichkeiten der Folgewörter.
3. Vergleiche: Was ändert sich, wenn das Modell zwei Wörter als Kontext benutzt statt einem?

Meine Rechnungen und mein Vergleich:

Gymnasium Ottweiler

Arbeitsauftrag 21: Ausgabe vorhersagen ●●○

Bestimme **ohne Computer**, was das Programm ausgibt. Wie viele Zeilen sind es?

```
korpus = "ich mag eis ich mag pizza"
woerter = korpus.split()
for i in range(len(woerter) - 2):
    print(woerter[i], woerter[i + 1], woerter[i + 2])
```

Ausgabe:

Arbeitsauftrag 22: Korpus gesucht ●●●

1. Schreibe einen möglichst kurzen Korpus, für den gilt:
   - Auf „ich“ folgt mit der Wahrscheinlichkeit $\frac{2}{3}$ „lache“ und mit $\frac{1}{3}$ „weine“.
   - Auf „lache“ folgt immer „ich“.
2. Kontrolliere deinen Korpus, indem du die Wahrscheinlichkeitstabelle aufstellst.
3. Begründe, warum ein solcher Korpus aus mindestens 6 Wörtern bestehen muss.

Mein Korpus, meine Kontrolle und meine Begründung:

Gymnasium Ottweiler

4.2.9 Das Wichtigste in Kürze

- Ein **n-Gramm** ist eine Folge von n aufeinanderfolgenden Wörtern: Unigramm (1), Bigramm (2), Trigramm (3). Ein Text mit N Wörtern hat $N - 1$ Bigramme.
- Der **Textkorpus** ist der Trainingstext. Bei der **Textkörperanalyse** zählt man, wie oft jedes n-Gramm darin vorkommt.
- Bigramme findet man mit einem „Fenster“ aus zwei Wörtern, das Wort für Wort weiterrückt. Gezählt wird mit einer Strichliste.
- $P(A \to B)$ = Anzahl von „A B“ geteilt durch die Anzahl aller Bigramme, die mit A beginnen.
- Für jedes Vorgängerwort ergeben die Wahrscheinlichkeiten zusammen 1 (100 %).
- Bei Trigrammen besteht der **Kontext** aus zwei Wörtern.
- In Python zerlegt `text.split()` einen Text in eine Liste von Wörtern. Damit kann man alle n-Gramme bilden und zählen.

Selbstcheck

- ☐ Ich kann zu einem Satz alle Unigramme, Bigramme und Trigramme angeben.
- ☐ Ich kann erklären, was ein Textkorpus und eine Textkörperanalyse sind.
- ☐ Ich kann in einem kurzen Text alle Bigramme von Hand zählen.
- ☐ Ich kann die Wahrscheinlichkeit für ein Folgewort berechnen und weiß, durch welche Zahl ich teilen muss.
- ☐ Ich kann eine Wahrscheinlichkeitstabelle aufstellen und mit der Summe 1 kontrollieren.
- ☐ Ich kann mit `split()` und einer Schleife alle Bigramme eines Textes ausgeben lassen.

Gymnasium Ottweiler

4.3 Textanalyse mit dem Computer

Leitidee

Zwei Sätze von Hand auszuzählen, geht gut. Aber ein echtes Sprachmodell lernt aus Millionen von Sätzen. Dort würdest du mit der Strichliste nie fertig. In diesem Abschnitt überlässt du das Zählen dem Computer: Du schreibst ein Python-Programm, das zu jedem Wort eines Textes die möglichen Folgewörter mit ihrer Häufigkeit ermittelt. Außerdem lernst du **SoekiaGPT** kennen, ein Lernprogramm im Browser, das dieselbe Analyse an größeren Textsammlungen vorführt.

Lernziele

Nach diesem Abschnitt kannst du

- begründen, warum größere Textkorpora rechnergestützt analysiert werden,
- beschreiben, wie n-Gramm-Häufigkeiten in einer Tabelle verwaltet werden,
- ein Python-Programm nutzen und erweitern, das zu einem Wort alle Folgewörter mit Häufigkeit und Wahrscheinlichkeit ermittelt,
- einen eigenen Textkorpus mit diesem Programm analysieren,
- mit SoekiaGPT n-Gramme in einer Textsammlung untersuchen.

4.3.1 Warum der Computer zählen muss

Im letzten Abschnitt hattest du Texte mit 9 bis 12 Wörtern. Schon da musstest du sorgfältig arbeiten, um kein Bigramm zu übersehen. Ein einziges Buch hat leicht 50 000 Wörter, also fast genauso viele Bigramme. Sprachmodelle werden mit Textsammlungen trainiert, die noch sehr viel größer sind.

Von Hand ist das nicht zu schaffen. Ein Programm erledigt dieselbe Arbeit dagegen in Sekundenbruchteilen und verzählt sich nie. Der Ablauf bleibt aber genau derselbe, den du schon kennst: Text in Wörter zerlegen, Bigramme bilden, zählen, Wahrscheinlichkeiten berechnen.

Merke

Bei größeren Textkorpora ist eine **Auszählung von Hand** nicht mehr praktikabel. Man nutzt Software, die die Häufigkeiten **automatisiert** ermittelt.

4.3.2 Ein größerer Korpus

Wir untersuchen einen Text aus fünf Zeilen über einen Hund und eine Katze. Damit er gut lesbar bleibt, soll er auch im Programm auf mehrere Zeilen verteilt sein.

Werkzeug

Einen Text über **mehrere Zeilen** schreibst du in Python zwischen **drei** Anführungszeichen `"""` am Anfang und am Ende. Die Zeilenumbrüche gehören dann zum Text dazu. Für `split()` ist ein Zeilenumbruch dasselbe wie ein Leerzeichen:

```python
text = """eins zwei
drei"""
print(text)
print(text.split())
```

AUSGABE

```
eins zwei
drei
['eins', 'zwei', 'drei']
```

4.3.3 Schritt 1: Alle Folgewörter eines Wortes sammeln

Von Hand hast du für ein Vorgängerwort alle Bigramme gesucht, die mit ihm beginnen, und dir jeweils das Folgewort notiert. Genau das macht die Funktion `folgewoerter`. Sie bekommt die Wortliste und ein Wort und liefert eine Liste mit **allen** Wörtern, die im Text direkt nach diesem Wort stehen – auch die doppelten.

```python
def folgewoerter(woerter, wort):
    folge = []
    for i in range(len(woerter) - 1):
        if woerter[i] == wort:
            folge.append(woerter[i + 1])
    return folge

korpus = """
der hund läuft im park der hund spielt mit dem ball
die katze sitzt auf dem baum die katze beobachtet den hund
der hund bellt laut der hund läuft zur katze
die katze springt vom baum die katze rennt weg
der hund läuft ihr hinterher der hund ist müde
"""
woerter = korpus.split()
print(len(woerter), "Wörter")
print(folgewoerter(woerter, "der"))
print(folgewoerter(woerter, "dem"))
```

AUSGABE

```
49 Wörter
['hund', 'hund', 'hund', 'hund', 'hund', 'hund']
['ball', 'baum']
```

- **Zeilen 1–6:** Die Funktion geht mit `i` alle Positionen durch, auf die noch ein Wort folgt. Steht an Position `i` das gesuchte Wort, wird das Wort an Position `i + 1` in die Liste `folge` aufgenommen. Das ist das „Fenster“ aus zwei Wörtern, nur dass jetzt bloß die Fenster beachtet werden, die mit `wort` beginnen.
- **Zeilen 8–14:** Der Korpus als mehrzeiliger Text.
- **Zeile 15:** `split()` zerlegt den Text in eine Liste mit 49 Wörtern. Also gibt es 48 Bigramme.
- **Zeilen 17–18:** Nach „der“ steht im ganzen Text sechsmal „hund“ und sonst nichts. Nach „dem“ steht einmal „ball“ und einmal „baum“.

Die Länge dieser Liste ist genau die Anzahl der Bigramme, die mit dem Wort beginnen. Das ist die Zahl, die du für die Wahrscheinlichkeit im **Nenner** brauchst. Auch der Sonderfall „letztes Wort“ ist automatisch richtig, denn die Schleife schaut nur Positionen an, auf die noch ein Wort folgt.

4.3.4 Schritt 2: Häufigkeiten zählen

Die Liste `['ball', 'baum']` ist noch keine Tabelle. Für die Tabelle brauchen wir jedes Folgewort **einmal** und daneben seine Anzahl – wie bei der Strichliste. Dafür gibt es zwei kleine Hilfsfunktionen, die du in ähnlicher Form schon geschrieben hast:

```
1  def ohne_doppelte(liste):
2      ergebnis = []
3      for element in liste:
4          if element not in ergebnis:
5              ergebnis.append(element)
6      return ergebnis
7
8  def zaehle(liste, gesucht):
9      anzahl = 0
10     for element in liste:
11         if element == gesucht:
12             anzahl += 1
13     return anzahl
14
15 def zeige_tabelle(woerter, wort):
16     folge = folgewoerter(woerter, wort)
17     print("Mögliche Folgewörter nach", wort)
18     for kandidat in ohne_doppelte(folge):
19         print(f"  {kandidat}: {zaehle(folge, kandidat)}-mal")
20
21 zeige_tabelle(woerter, "der")
22 zeige_tabelle(woerter, "katze")
```

AUSGABE

```
Mögliche Folgewörter nach der
  hund: 6-mal
Mögliche Folgewörter nach katze
  sitzt: 1-mal
  beobachtet: 1-mal
  die: 1-mal
  springt: 1-mal
  rennt: 1-mal
```

- **Zeilen 1–6:** `ohne_doppelte` liefert eine neue Liste, in der jedes Element nur einmal vorkommt, und zwar in der Reihenfolge seines ersten Auftretens. Das sind die **Kandidaten** für das nächste Wort.
- **Zeilen 8–13:** `zaehle` zählt, wie oft `gesucht` in der Liste steht.
- **Zeilen 15–19:** `zeige_tabelle` setzt alles zusammen: Erst werden die Folgewörter gesammelt, dann wird für jeden Kandidaten die Anzahl ausgegeben.
- **Zeilen 21–22:** Nach „der“ folgt immer „hund“. Nach „katze“ folgen fünf verschiedene Wörter, jedes genau einmal. Dass auch „die“ dabei ist, liegt an „zur katze die katze“: Zwischen der dritten und vierten Zeile des Korpus gibt es keine Grenze, `split()` macht aus allem eine einzige Wortliste.

Stolperstein

Das Programm weiß nichts von Sätzen. Für die Analyse ist der Korpus eine lange Kette von Wörtern. Deshalb entstehen auch Bigramme über das Satzende hinweg, etwa „ball die“ oder „müde“ ganz am Ende ohne Nachfolger. Echte Sprachmodelle behandeln Satzzeichen und Satzgrenzen genauer. Für unser Modell vereinfachen wir das bewusst.

Versuch: Probiere es aus

Rufe `zeige_tabelle` für das Wort „müde“ auf. Warum wird nur die Überschrift ausgegeben? Rufe die Funktion auch mit einem Wort auf, das im Text gar nicht vorkommt, zum Beispiel „maus“.

4.3.5 Die Bigramm-Tabelle

Wenn man `zeige_tabelle` für jedes Wort des Korpus aufruft, entsteht die vollständige **Bigramm-Tabelle** des Modells. Ein Ausschnitt:

Kontext	mögliche Folgewörter (Anzahl)
der	hund (6)
die	katze (4)
hund	?
katze	sitzt (1), beobachtet (1), die (1), springt (1), rennt (1)
läuft	im (1), zur (1), ihr (1)
dem	ball (1), baum (1)
baum	die (2)

Das Wort in der ersten Spalte, nach dem die Folgewörter gesucht werden, heißt **Kontext**. Bei einem Bigramm-Modell ist der Kontext ein einzelnes Wort, bei einem Trigramm-Modell ein Wortpaar.

So eine Tabelle ist eine **Tabelle von Tabellen**: Zu jedem Kontext gehört eine kleine eigene Tabelle aus Folgewort und Anzahl. Man kann sie sich wie ein **Wörterbuch** vorstellen: Unter dem Stichwort „dem“ schlägt man nach und findet dort „ball: 1, baum: 1“.

In vielen Programmiersprachen gibt es genau dafür einen eigenen Datentyp, das **Dictionary** (englisch für Wörterbuch). In der Python-Schreibweise sieht ein Ausschnitt der Tabelle so aus:

```
{"der":  {"hund": 6},
 "dem":  {"ball": 1, "baum": 1},
 "baum": {"die": 2}}
```

Vor jedem Doppelpunkt steht ein Stichwort, dahinter der Eintrag dazu. Der Eintrag ist selbst wieder ein kleines Wörterbuch. Deshalb spricht man von einem **Dictionary von Dictionaries**.

Merke

- Die Ergebnisse werden meist **tabellarisch** verwaltet: Zu jedem **Kontext** (z. B. einem Wort oder Wortpaar) wird eine Liste möglicher Folgewörter mit ihrer jeweiligen Häufigkeit bzw. Wahrscheinlichkeit gespeichert.
- Eine gängige Datenstruktur dafür ist ein **Dictionary von Dictionaries**: `{kontext: {folgewort: anzahl}}`.

Tipp

Dictionaries brauchst du in diesem Skript nicht selbst zu programmieren. Unser Programm kommt mit Listen aus: Es speichert nicht die ganze Tabelle, sondern berechnet die Zeile für ein Wort jedes Mal neu, wenn sie gebraucht wird. Das ist etwas langsamer, liefert aber genau dieselben Zahlen.

Arbeitsauftrag 23: Eigene Analyse

Gib mit dem Programm für die Wörter „die“ und „hund“ jeweils aus, welche Folgewörter mit welcher Häufigkeit vorkommen. Ergänze damit die fehlende Zeile „hund“ in der Tabelle oben.

Ergebnis:

4.3.6 Schritt 3: Wahrscheinlichkeiten berechnen

Für die Wahrscheinlichkeit teilst du die Anzahl eines Folgewortes durch die Anzahl **aller** Bigramme, die mit dem Kontext beginnen. Diese Zahl ist die Länge der Liste `folgewoerter(woerter, wort)`. Für „hund“ gilt zum Beispiel $P(\text{hund} \rightarrow \text{läuft}) = \frac{3}{7} \approx 0{,}43$.

Das Programm soll das für alle Kandidaten eines Wortes ausrechnen. Dabei entstehen **zwei Listen, die zusammengehören** wie die zwei Spalten einer Tabelle: eine Liste mit den Kandidaten und eine mit ihren Wahrscheinlichkeiten. Das Element an Stelle 0 der einen Liste gehört zum Element an Stelle 0 der anderen Liste, und so weiter.

Index	**0**	**1**	**2**	**3**	**4**
Kandidaten	läuft	spielt	der	bellt	ist
Wahrscheinlichkeiten	?	?	?	?	?

Arbeitsauftrag 24: Wahrscheinlichkeiten berechnen

Schreibe eine Funktion `wahrscheinlichkeiten(woerter, wort)`, die für ein gegebenes Wort eine Liste mit den Wahrscheinlichkeiten seiner Folgewörter zurückgibt, und zwar in derselben Reihenfolge wie `ohne_doppelte(folgewoerter(woerter, wort))`. Die Wahrscheinlichkeit ist die Anzahl des Folgewortes geteilt durch die Gesamtanzahl aller Folgewörter zu diesem Wort. Runde auf zwei Nachkommastellen.

Teste die Funktion mit „der“, „die“ und „hund“ und fülle damit die Tabelle oben aus.

4.3.7 SoekiaGPT: Textanalyse im Browser

Für das Arbeiten mit größeren Textsammlungen gibt es Lernprogramme, die dir die n-Gramme und ihre Häufigkeiten anschaulich zeigen. Der Lehrplan empfiehlt **SoekiaGPT**. Es läuft im Browser: www.soekia.ch/GPT.

In SoekiaGPT kannst du eine vorgegebene **Textsammlung** auswählen (zum Beispiel Märchen); je nach Version kannst du auch eigene Texte hinzufügen. Das Programm analysiert die Sammlung und zeigt dir die gefundenen **n-Gramme mit ihren Häufigkeiten**. Gibst du einen Satzanfang ein, zeigt es dir die möglichen **Folgewörter** und setzt den Text Wort für Wort fort. Die genaue Gestaltung der Oberfläche kann sich von Version zu Version ändern. Lass dich davon nicht verwirren: Das Prinzip ist dasselbe wie in deinem Python-Programm.

Beide Zugänge ergänzen sich: SoekiaGPT zeigt das Verfahren anschaulich an größeren, echten Textsammlungen. Dein Python-Programm zeigt dir, was dabei „unter der Haube“ passiert.

Arbeitsauftrag 25: SoekiaGPT erkunden

Öffne SoekiaGPT im Browser.

1. Wähle eine der vorgegebenen Textsammlungen aus. Suche die Anzeige der n-Gramme und notiere drei Bigramme, die besonders häufig vorkommen.
2. Gib einen kurzen Satzanfang ein, der zur Textsammlung passt, und lass dir mögliche Folgewörter anzeigen. Notiere den Satzanfang und zwei vorgeschlagene Folgewörter.
3. Beschreibe in zwei bis drei Sätzen, welche Schritte deines Python-Programms du in SoekiaGPT wiedererkennst.

Meine Beobachtungen:

Gymnasium Ottweiler

4.3.8 Übungen

Arbeitsauftrag 26: Eigenen Textkorpus analysieren

Suche dir einen eigenen, etwas längeren Text: ein paar Sätze aus einem Buch, eine Zusammenfassung eines Liedtextes in eigenen Worten oder etwas frei Erfundenes. Schreibe ihn klein und ohne Satzzeichen in einen mehrzeiligen Text und analysiere ihn mit den Funktionen aus diesem Abschnitt.

Welches Wort hat die **meisten unterschiedlichen** Folgewörter? Schreibe ein Programm, das dieses Wort findet.

Tipp

So kannst du vorgehen: Gehe mit einer Schleife über alle **verschiedenen** Wörter des Textes. Berechne für jedes Wort, wie viele unterschiedliche Folgewörter es hat. Das größte Ergebnis findest du wie das Maximum in einer Liste. Der Anfang könnte so aussehen:

```
for wort in ohne_doppelte(meine_woerter):
    anzahl = len(ohne_doppelte(folgewoerter(meine_woerter, wort)))
```

4.3.9 Weitere Übungen

Hier kannst du das Gelernte weiter üben. Die Punkte im Titel zeigen, wie schwer eine Aufgabe ist: ●○○ leicht, ●●○ mittel, ●●● knifflig. Fang mit den leichten an.

Für die Programmieraufgaben brauchst du die Funktionen `folgewoerter`, `ohne_doppelte` und `zaehle` und die Wortliste `woerter` des Hund-Katze-Korpus aus diesem Abschnitt. Schreibe deine Programme in derselben Datei darunter.

Arbeitsauftrag 27: Warum der Computer zählt ●○○

1. Ein Jugendbuch hat 60 000 Wörter. Bestimme, wie viele Bigramme es enthält.
2. Nenne zwei Gründe, warum man die Bigramme eines solchen Buches nicht von Hand auszählt.

Meine Antworten:

Gymnasium Ottweiler

Arbeitsauftrag 28: Eine Tabelle lesen ●○○

Ein Programm hat einen Korpus analysiert. Ein Ausschnitt seiner Bigramm-Tabelle in der Schreibweise `{kontext: {folgewort: anzahl}}`:

```
{"ich":   {"gehe": 3, "bleibe": 1},
 "gehe":  {"heute": 2, "morgen": 1},
 "heute": {"ins": 2}}
```

1. Nenne alle Kontexte in diesem Ausschnitt.
2. Nenne die Folgewörter von „gehe“ mit ihren Häufigkeiten.
3. Wie viele Bigramme beginnen mit „ich“?
4. Berechne die Wahrscheinlichkeit, dass auf „ich“ das Wort „gehe“ folgt.
5. Mit welcher Wahrscheinlichkeit folgt auf „heute“ das Wort „ins“?

Meine Antworten:

Arbeitsauftrag 29: Was liefert folgewoerter? ●○○

Bestimme **ohne Computer** die Ausgabe des Programms. Erkläre dann, warum zwei der Listen leer sind, und zwar aus verschiedenen Gründen.

```
1 schule = "wir lernen mathe wir lernen englisch wir chillen"
2 woerter_schule = schule.split()
3 print(folgewoerter(woerter_schule, "wir"))
4 print(folgewoerter(woerter_schule, "lernen"))
5 print(folgewoerter(woerter_schule, "chillen"))
6 print(folgewoerter(woerter_schule, "schule"))
```

Ausgabe und Erklärung:

Arbeitsauftrag 30: Eine Tabelle aufschreiben ●●○

Gegeben ist der Textkorpus:

> tom spielt fußball lea spielt tennis tom spielt tennis

1. Stelle die vollständige Bigramm-Tabelle in der Schreibweise `{kontext: {folgewort: anzahl}}` auf.
2. Berechne die Wahrscheinlichkeiten aller Folgewörter von „spielt“.

Meine Tabelle und meine Rechnung:

Gymnasium Ottweiler

Arbeitsauftrag 31: Fehler finden ●●○

Ben hat die Funktion `folgewoerter` aus dem Gedächtnis aufgeschrieben. Beim ersten Aufruf klappt alles, beim zweiten nicht:

```python
def folgewoerter_ben(woerter, wort):
    folge = []
    for i in range(len(woerter)):
        if woerter[i] == wort:
            folge.append(woerter[i + 1])
    return folge

test = "der hund bellt der hund schläft".split()
print(folgewoerter_ben(test, "der"))
print(folgewoerter_ben(test, "schläft"))
```

AUSGABE

```
['hund', 'hund']
```

FEHLERMELDUNG

```
IndexError: list index out of range
```

1. Erkläre, warum der erste Aufruf funktioniert und der zweite zu einem Fehler führt.
2. Beschreibe, wie Ben den Fehler behebt.

Meine Erklärung:

Arbeitsauftrag 32: Das häufigste Wort ●●○

Bei der Textkörperanalyse kann man auch **Unigramme** zählen, also einzelne Wörter. Schreibe ein Programm, das im Hund-Katze-Korpus

1. ausgibt, wie viele verschiedene Wörter der Korpus enthält,
2. das Wort findet, das am häufigsten vorkommt, und es mit seiner Anzahl ausgibt.

Verwende `ohne_doppelte` und `zaehle`.

Tipp

Gehe mit einer Schleife über `ohne_doppelte(woerter)` und merke dir das bisher häufigste Wort und seine Anzahl – wie bei der Suche nach dem Maximum.

Arbeitsauftrag 33: Wahrscheinlichkeit für ein Wortpaar ●●●

Schreibe eine Funktion `wahrscheinlichkeit(woerter, wort, folgewort)`, die die Wahrscheinlichkeit zurückgibt, dass auf `wort` das Wort `folgewort` folgt, gerundet auf zwei Nachkommastellen. Hat `wort` keine Folgewörter (es kommt nicht vor oder nur als letztes Wort), soll die Funktion `0` zurückgeben.

Teste mit dem Hund-Katze-Korpus:

- „hund“ → „läuft“ (erwartet: $\frac{3}{7} \approx 0{,}43$),
- „dem“ → „baum“,
- „katze“ → „bellt“,
- „maus“ → „läuft“ und „müde“ → „ist“.

4.3.10 Das Wichtigste in Kürze

- Große Textkorpora werden nicht von Hand, sondern **automatisiert** mit Software analysiert.
- `folgewoerter(woerter, wort)` sammelt alle Wörter, die direkt nach `wort` stehen. Die Länge dieser Liste ist der Nenner für die Wahrscheinlichkeit.
- Mit `ohne_doppelte` und `zaehle` entsteht daraus eine Zeile der **Bigramm-Tabelle**: Kandidaten mit ihren Häufigkeiten.
- Das Wort, nach dessen Folgewörtern gesucht wird, heißt **Kontext**.
- Die Ergebnisse werden **tabellarisch** verwaltet. Eine gängige Datenstruktur dafür ist ein **Dictionary von Dictionaries** `{kontext: {folgewort: anzahl}}`. Unser Programm bildet die Tabelle mit Listen nach.
- **SoekiaGPT** zeigt n-Gramme, ihre Häufigkeiten und mögliche Folgewörter an größeren Textsammlungen.

Selbstcheck

- ☐ Ich kann begründen, warum man große Textkorpora mit dem Computer analysiert.
- ☐ Ich kann erklären, was die Funktion `folgewoerter` macht und warum ihre Länge der Nenner der Wahrscheinlichkeit ist.
- ☐ Ich kann mit dem Programm die Folgewörter eines Wortes mit Häufigkeit und Wahrscheinlichkeit ausgeben.
- ☐ Ich kann beschreiben, wie eine Bigramm-Tabelle aufgebaut ist und was „Kontext“ bedeutet.
- ☐ Ich kann einen eigenen Text mit dem Programm untersuchen.
- ☐ Ich kann in SoekiaGPT eine Textsammlung auswählen und mir n-Gramme und Folgewörter anzeigen lassen.

4.4 Texte erzeugen: next word prediction und Sampling

Leitidee

Bisher hast du ein Sprachmodell nur **trainiert**: Du hast gezählt und Wahrscheinlichkeiten berechnet. Jetzt lässt du es **schreiben**. Die Idee ist verblüffend einfach: Das Modell schaut auf das letzte Wort, wählt mithilfe seiner Tabelle ein passendes nächstes Wort aus, hängt es an – und wiederholt das immer wieder. Spannend ist dabei die Frage, **wie** es unter mehreren möglichen Folgewörtern auswählt. Davon hängt ab, ob der Text eintönig, abwechslungsreich oder unsinnig wird. Auch die Trainingstexte und die Länge der n-Gramme verändern das Ergebnis deutlich.

Lernziele

Nach diesem Abschnitt kannst du

- beschreiben, wie ein n-Gramm-Modell Wort für Wort einen Text erzeugt (next word prediction),
- die Sampling-Strategien greedy, random und top-k benennen und von Hand anwenden,
- Textgenerierung mit greedy, random und top-k in Python umsetzen und vergleichen,
- beschreiben, wie sich die Länge der n-Gramme auf den erzeugten Text auswirkt,
- erläutern, wie der erzeugte Text von den Trainingsdaten abhängt.

4.4.1 Vom Modell zum Text

Ein Text entsteht beim n-Gramm-Modell Wort für Wort. Man gibt ein **Startwort** vor. Dann wiederholt das Modell immer dieselben drei Schritte:

1. Schaue in der Tabelle nach, welche Wörter auf das **letzte Wort** folgen können.
2. **Wähle** eines dieser Folgewörter aus.
3. Hänge es an den Text an. Es ist jetzt das neue letzte Wort.

Das Modell sagt also immer nur das **nächste Wort** voraus. Auf Englisch heißt das **next word prediction**. Hat das letzte Wort keine Folgewörter (weil es im Korpus nur ganz am Ende steht), bricht das Verfahren ab.

Definition: Textgenerierung (next word prediction)

Textgenerierung ist das Erzeugen eines neuen Textes durch ein Sprachmodell. Bei der **next word prediction** (Vorhersage des nächsten Wortes) bestimmt das Modell aus dem bisherigen Text ein passendes nächstes Wort.

Merke

Textgenerierung (next word prediction): Ausgehend von einem Startwort bzw. Startkontext wird mithilfe der n-Gramm-Tabelle immer wieder das **nächste Wort** bestimmt und an den Text angehängt.

Die naheliegendste Art, ein Folgewort auszuwählen: Man nimmt immer das **wahrscheinlichste**. Probiere das zuerst von Hand aus.

Arbeitsauftrag 34: Von Hand: die ersten Schritte der Textgenerierung

Verwende die Bigramm-Tabelle zum Hund-Katze-Korpus aus dem Abschnitt *Textanalyse mit dem Computer* („der hund läuft im park der hund spielt mit dem ball …“). Die Zeilen, die du brauchst, findest du dort in der Tabelle und in deinem Ergebnis zum Auftrag *Eigene Analyse* (Zeile „hund“). Für „im“ und „park“ gilt: Auf „im“ folgt immer „park“, auf „park“ immer „der“.

Beginne beim Wort „der“. Wähle jeweils das **wahrscheinlichste** Folgewort, bis du **5 Wörter** notiert hast. Haben mehrere Folgewörter die gleiche Wahrscheinlichkeit, nimm das, das in der Tabelle zuerst steht.

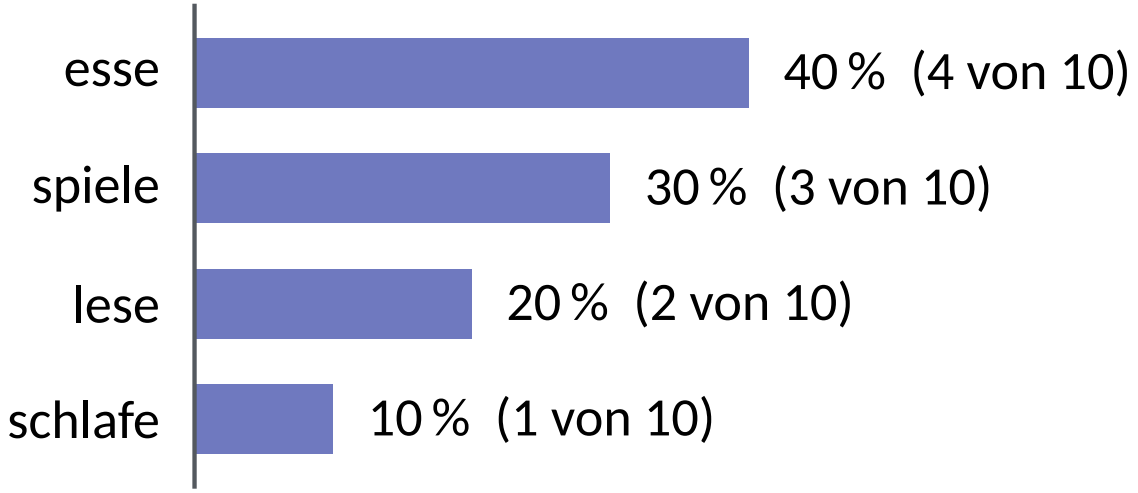
Mein Text aus 5 Wörtern:

4.4.2 Sampling: Wie wird das Folgewort ausgewählt?

Oft gibt es mehrere mögliche Folgewörter. Die Art, wie das Modell eines davon auswählt, heißt **Sampling-Strategie** (englisch *sample* = Stichprobe, Probe ziehen). Wir betrachten drei Strategien an einem kleinen Korpus über das Wochenende:

```
ich esse pizza     ich spiele fußball   ich esse eis
ich lese comics    ich spiele schach    ich esse pizza
ich schlafe lange  ich lese bücher      ich spiele fußball
ich esse nudeln
```

Das Wort „ich“ kommt zehnmal vor, und immer folgt ein Wort: viermal „esse“, dreimal „spiele“, zweimal „lese“ und einmal „schlafe“. Daraus ergeben sich diese Wahrscheinlichkeiten:



Mögliche Folgewörter nach „ich“ im Wochenend-Korpus mit ihren Wahrscheinlichkeiten

Definition: Sampling

Sampling ist die Auswahl des nächsten Wortes aus den möglichen Folgewörtern. Die **Sampling-Strategie** legt fest, **wie** ausgewählt wird.

Greedy: immer das wahrscheinlichste Wort

Bei der Strategie **greedy** (englisch für „gierig“) nimmt das Modell immer das Wort mit dem längsten Balken. Nach „ich“ kommt also **jedes Mal** „esse“. Der Name passt: Das Modell greift immer sofort nach dem größten Stück, ohne an später zu denken. Den Begriff „gierig“ kennst du vielleicht schon vom Problemlösen mit Heuristiken.

Greedy ist **deterministisch**: Gleiches Startwort, gleicher Text – jedes Mal. Das hat einen Nachteil. Im Wochenend-Korpus folgt auf „esse“ am häufigsten „pizza“, auf „pizza“ immer „ich“, und auf „ich“ wieder „esse“. Der Text dreht sich im Kreis: „ich esse pizza ich esse pizza ich esse pizza …“. Die Wörter „spiele“, „lese“ und „schlafe“ kommen nie vor, obwohl sie zusammen sogar häufiger sind als „esse“.

Random: Auslosen nach Wahrscheinlichkeit

Bei der Strategie **random** (englisch für „zufällig“) wird das Folgewort **ausgelost**. Dabei bekommt aber nicht jedes Wort die gleiche Chance. Häufige Wörter sollen öfter gezogen werden als seltene. Man sagt: Die Auswahl ist **gewichtet** nach der Wahrscheinlichkeit.

Das kannst du dir wie eine **Lostrommel** vorstellen. Für jedes Bigramm im Korpus kommt ein Los hinein. Nach „ich“ sind das 10 Lose: 4 mit „esse“, 3 mit „spiele“, 2 mit „lese“ und 1 mit „schlafe“. Wer blind ein Los zieht, erwischt „esse“ mit der Wahrscheinlichkeit $\frac{4}{10} = 40\,\%$ – genau wie in der Tabelle.



Die Lostrommel für „ich“: zehn Lose, nummeriert von 1 bis 10. Eine Zufallszahl von 1 bis 10 bestimmt das Los

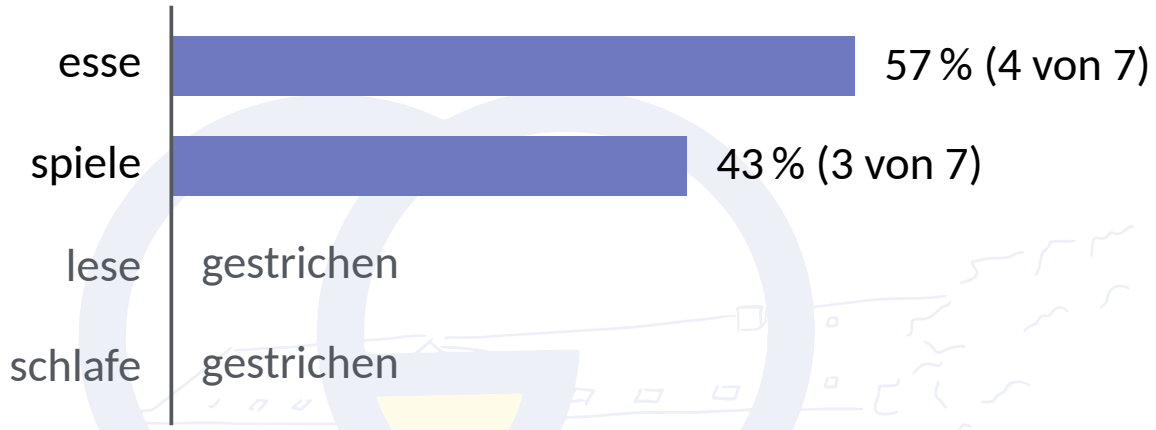
Mit einem Zehnerwürfel oder einem Zufallszahlengenerator geht das auch ohne Trommel: Ziehe eine Zufallszahl von 1 bis 10 und schau nach, welches Wort auf diesem Los steht. Die Zahlen 1 bis 4 bedeuten „esse“, 5 bis 7 „spiele“, 8 und 9 „lese“ und 10 „schlafe“.

Random ist **nicht deterministisch**: Jeder Durchlauf kann einen anderen Text liefern. Die Texte werden abwechslungsreicher. Aber auch seltene Folgewörter kommen manchmal dran, und dann kann der Text seltsam werden.

Top-k: nur unter den besten k auslosen

Die Strategie **top-k** ist ein Kompromiss. Zuerst werden nur die k wahrscheinlichsten Folgewörter behalten, alle anderen fliegen raus. Unter den übrigen wird dann wie bei random gewichtet ausgelost. Die Zahl k legt man vorher fest.

Bei **top-2** bleiben nach „ich“ nur „esse“ und „spiele“ übrig. In der Lostrommel liegen jetzt nur noch 4 + 3 = 7 Lose. Die Wahrscheinlichkeiten ändern sich deshalb: „esse“ hat $\frac{4}{7} \approx 57\,\%$ und „spiele“ $\frac{3}{7} \approx 43\,\%$.



Top-2 nach „ich“: „lese“ und „schlafe“ werden gestrichen. Unter „esse“ und „spiele“ wird mit 7 Losen neu gelost

Mit $k = 1$ bleibt nur das wahrscheinlichste Wort übrig – das ist greedy. Ist k mindestens so groß wie die Zahl aller Folgewörter, wird nichts gestrichen – das ist random. Top-k liegt also **zwischen** den beiden anderen Strategien.

Merke

Sampling-Strategien legen fest, **wie** aus mehreren möglichen Folgewörtern eines ausgewählt wird:

- **Greedy:** Es wird **immer** das **wahrscheinlichste** Folgewort gewählt → deterministisch, aber oft eintönig und sich wiederholend.
- **Random:** Es wird **zufällig** unter allen möglichen Folgewörtern gewählt, **gewichtet nach ihrer Wahrscheinlichkeit** → abwechslungsreicher, aber manchmal unsinnig.
- **Top-k:** Es werden nur die k wahrscheinlichsten Folgewörter betrachtet, unter denen dann zufällig (gewichtet) ausgewählt wird → Kompromiss zwischen Vorhersagbarkeit und Abwechslung.

So kannst du es sagen

So beschreibst du die Strategien in Fachsprache:

- *„Greedy nimmt das Beste.“ → „Bei greedy wird stets das Folgewort mit der höchsten Wahrscheinlichkeit gewählt.“*
- *„Random würfelt.“ → „Bei random wird das Folgewort zufällig ausgewählt, wobei häufigere Folgewörter mit höherer Wahrscheinlichkeit gezogen werden.“*

Arbeitsauftrag 35: Sampling von Hand

Verwende die Folgewörter nach „ich“ aus dem Wochenend-Korpus (4 × esse, 3 × spiele, 2 × lese, 1 × schlafe).

1. Welches Wort wählt greedy?
2. Bei random werden nacheinander die Zufallszahlen 7, 2, 10 und 9 gezogen (Lose wie in der Abbildung). Welche Wörter werden gewählt?
3. Bei top-2 sind die Lose 1 bis 4 „esse“ und 5 bis 7 „spiele“. Welche Wörter werden bei den Zufallszahlen 5 und 3 gewählt?
4. Berechne die Wahrscheinlichkeiten der Folgewörter bei **top-3** und zeichne ein Balkendiagramm.

Meine Antworten und mein Balkendiagramm:

4.4.3 Textgenerierung in Python

Jetzt bringen wir das alles in ein Programm. Wir brauchen die Funktionen aus dem Abschnitt *Textanalyse mit dem Computer*. Sie stehen hier noch einmal zusammen mit dem Hund-Katze-Korpus:

```python
def folgewoerter(woerter, wort):
    folge = []
    for i in range(len(woerter) - 1):
        if woerter[i] == wort:
            folge.append(woerter[i + 1])
    return folge

def ohne_doppelte(liste):
    ergebnis = []
    for element in liste:
        if element not in ergebnis:
            ergebnis.append(element)
    return ergebnis

def zaehle(liste, gesucht):
    anzahl = 0
    for element in liste:
        if element == gesucht:
            anzahl += 1
    return anzahl

korpus = """
der hund läuft im park der hund spielt mit dem ball
die katze sitzt auf dem baum die katze beobachtet den hund
der hund bellt laut der hund läuft zur katze
die katze springt vom baum die katze rennt weg
der hund läuft ihr hinterher der hund ist müde
"""
woerter = korpus.split()
```

Dieses Programm gibt noch nichts aus. Es stellt nur die Werkzeuge und die Wortliste bereit. Alle folgenden Programme in diesem Abschnitt bauen darauf auf: Schreibe sie in derselben Datei darunter.

Greedy in Python

Die Funktion für greedy sucht unter den Kandidaten den mit der größten Anzahl. Das ist dieselbe Idee wie die Suche nach dem Maximum in einer Liste:

```python
def naechstes_wort_greedy(woerter, wort):
    folge = folgewoerter(woerter, wort)
    if len(folge) == 0:
        return ""
    bestes = ""
    beste_anzahl = 0
    for kandidat in ohne_doppelte(folge):
        anzahl = zaehle(folge, kandidat)
        if anzahl > beste_anzahl:
            bestes = kandidat
            beste_anzahl = anzahl
    return bestes

print(naechstes_wort_greedy(woerter, "hund"))
print(naechstes_wort_greedy(woerter, "läuft"))
print(naechstes_wort_greedy(woerter, "müde"), "<- leer")
```

AUSGABE
```
läuft
im
 <- leer
```

- **Zeilen 2–4:** Hat das Wort keine Folgewörter, gibt die Funktion die leere Zeichenkette `""` zurück. Daran erkennt das aufrufende Programm später, dass es nicht weitergeht.
- **Zeilen 5–11:** Die Kandidaten werden der Reihe nach durchgegangen. Nur wenn ein Kandidat **mehr** Vorkommen hat als der bisher beste, wird er der neue beste. Bei Gleichstand bleibt der frühere Kandidat. Deshalb gewinnt nach „läuft“ das Wort „im“: Es steht unter den gleich häufigen Wörtern im Text zuerst.
- **Zeile 16:** „müde“ ist das letzte Wort des Korpus. Die Funktion gibt `""` zurück, deshalb steht vor dem Pfeil nichts.

Random in Python: die Lostrommel

Für random brauchen wir die Lostrommel. Und die haben wir schon! Die Liste `folgewoerter(woerter, wort)` enthält jedes Folgewort **so oft, wie es im Text vorkommt** – also genau ein Los pro Bigramm. Wir müssen nur eine zufällige Stelle dieser Liste auswählen:

```
1  import random
2
3  def naechstes_wort_random(woerter, wort):
4      folge = folgewoerter(woerter, wort)
5      if len(folge) == 0:
6          return ""
7      nummer = random.randint(0, len(folge) - 1)
8      return folge[nummer]
9
10 print(folgewoerter(woerter, "hund"))
11 print(naechstes_wort_random(woerter, "hund"))
12 print(naechstes_wort_random(woerter, "hund"))
13 print(naechstes_wort_random(woerter, "hund"))
```

AUSGABE
```
['läuft', 'spielt', 'der', 'bellt', 'läuft', 'läuft', 'ist']
läuft
der
läuft
```

Die Ausgabe ab der zweiten Zeile ist nur ein **Beispiel**. Bei dir kommen andere Wörter heraus, und bei jedem Start andere.

- **Zeile 1:** Das Modul `random` wird eingebunden.
- **Zeile 7:** `random.randint(0, len(folge) - 1)` liefert eine zufällige ganze Zahl von 0 bis zum letzten Index der Liste, jede mit gleicher Wahrscheinlichkeit. Das ist das blinde Ziehen eines Loses.
- **Zeile 8:** Das Wort an dieser Stelle wird zurückgegeben. Weil „läuft“ dreimal in der Liste steht, wird es mit der Wahrscheinlichkeit $\frac{3}{7}$ gezogen, jedes andere Wort mit $\frac{1}{7}$. Die Auswahl ist also **gewichtet**, ohne dass wir die Wahrscheinlichkeiten ausrechnen müssen.

Versuch: Probiere es aus: Stimmen die Wahrscheinlichkeiten?

Mit dem folgenden Programm (unter die bisherigen Programme schreiben) lässt du 1000-mal ein Folgewort nach „hund“ auslosen und zählst, wie oft „läuft“ kam. Erwartet sind etwa $\frac{3}{7} \cdot 1000 \approx 429$ Treffer. Starte es mehrmals.

```python
treffer = 0
for versuch in range(1000):
    if naechstes_wort_random(woerter, "hund") == "läuft":
        treffer += 1
print(f"läuft wurde {treffer}-mal gezogen")
```

AUSGABE

```
läuft wurde 437-mal gezogen
```

Die Zahl schwankt von Start zu Start (hier ein Beispiel), liegt aber fast immer in der Nähe von 429.

Einen ganzen Text erzeugen

Jetzt fehlt nur noch die Schleife, die Wort für Wort anhängt. Die Funktion `generiere_text` bekommt die Wortliste, ein Startwort, die gewünschte Länge (Anzahl der Wörter) und den Namen der Strategie als Zeichenkette:

```python
def generiere_text(woerter, startwort, laenge, strategie):
    text = startwort
    aktuelles_wort = startwort
    for schritt in range(laenge - 1):
        if strategie == "greedy":
            naechstes = naechstes_wort_greedy(woerter, aktuelles_wort)
        else:
            naechstes = naechstes_wort_random(woerter, aktuelles_wort)
        if naechstes == "":
            break
        text = text + " " + naechstes
        aktuelles_wort = naechstes
    return text

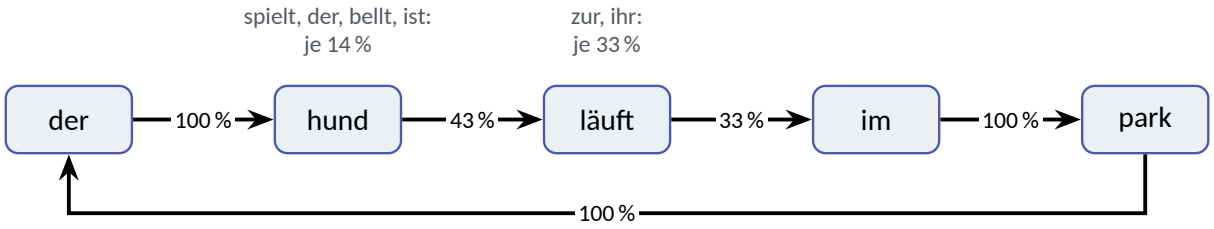
print("Greedy:", generiere_text(woerter, "der", 10, "greedy"))
```

AUSGABE

```
Greedy: der hund läuft im park der hund läuft im park
```

- **Zeilen 2–3:** Der Text beginnt mit dem Startwort. `aktuelles_wort` ist das jeweils letzte Wort.
- **Zeile 4:** Das Startwort ist schon da, also fehlen noch `laenge - 1` Wörter.
- **Zeilen 5–8:** Je nach Strategie wird das nächste Wort mit greedy oder random bestimmt.
- **Zeilen 9–10:** Gibt es kein Folgewort, wird die Schleife mit `break` vorzeitig beendet. Der Text ist dann kürzer als gewünscht.
- **Zeilen 11–12:** Das neue Wort wird mit einem Leerzeichen angehängt und wird zum aktuellen Wort.

Die ersten fünf Wörter stimmen mit deinem Ergebnis aus dem Auftrag *Von Hand: die ersten Schritte der Textgenerierung* überein. Danach beginnt greedy wieder von vorn: Der Text läuft in einer Schleife, wie die Abbildung zeigt.



Greedy im Hund-Katze-Korpus: Der Weg mit den jeweils höchsten Wahrscheinlichkeiten führt im Kreis. Die anderen Folgewörter (grau) werden nie gewählt

Und so sieht es mit random aus. Weil der Zufall mitspielt, sind die folgenden Ausgaben nur **Beispiele**:

```
print("Random:", generiere_text(woerter, "der", 10, "random"))
print("Random:", generiere_text(woerter, "der", 10, "random"))
print("Random:", generiere_text(woerter, "der", 10, "random"))
```

AUSGABE

```
Random: der hund läuft zur katze rennt weg der hund läuft
Random: der hund spielt mit dem baum die katze sitzt auf
Random: der hund ist müde
```

Im zweiten Beispiel ist ein neuer Satz entstanden, der so nicht im Korpus steht: „der hund spielt mit dem baum“. Jedes einzelne Wortpaar kommt im Korpus vor, die Kombination aber nicht. Im dritten Beispiel bricht der Text nach vier Wörtern ab, weil „müde“ keine Folgewörter hat.

Stolperstein

Wenn `generiere_text` einen `NameError` meldet, etwa `name 'naechstes_wort_random' is not defined`, fehlt eine der Funktionen in deiner Datei – oder sie steht **unter** dem Aufruf. Python muss eine Funktion kennen, bevor sie aufgerufen wird. Schreibe alle Funktionen an den Anfang der Datei.

4.4.4 Übungen zum Sampling

Arbeitsauftrag 36: Greedy und random vergleichen

Führe das Programm mit den drei Random-Zeilen und der Greedy-Zeile **mehrfach** aus. Was fällt dir beim Vergleich der Greedy- und der Random-Ausgaben auf? Erkläre, warum greedy immer denselben Text liefert, random aber unterschiedliche.

Meine Beobachtung und Erklärung:

Arbeitsauftrag 37 (Zusatz): Top-k-Sampling implementieren

Implementiere eine Funktion `naechstes_wort_top_k(woerter, wort, k)`, die nur unter den `k` häufigsten Folgewörtern zufällig auswählt, gewichtet nach ihrer Häufigkeit. Teste sie mit „hund“ und $k = 2$. Schreibe außerdem eine Funktion `generiere_text_top_k(woerter, startwort, laenge, k)`, die damit einen Text erzeugt.

Tipp

Ein möglicher Weg:

1. Lege zwei Listen an, `kandidaten` (ohne Doppelte) und `anzahlen`, die zusammengehören wie die zwei Spalten einer Tabelle.
2. Wiederhole k-mal: Suche die Position des größten Wertes in `anzahlen`. Nimm das Wort und seine Anzahl mit `pop` aus beiden Listen heraus und lege so viele Lose dieses Wortes in eine neue Liste `lostrommel`.
3. Ziehe zum Schluss ein zufälliges Los aus `lostrommel`.

4.4.5 Wie die Länge der n-Gramme den Text verändert

Bisher hat unser Modell nur auf **ein** Wort zurückgeschaut. Das ist der Grund für Sätze wie „der hund spielt mit dem baum“: Nach „dem“ weiß das Modell nicht mehr, dass vorher „spielt mit“ stand, und wählt „baum“ genauso oft wie „ball“.

Ein **Trigramm-Modell** schaut auf die letzten **zwei** Wörter. Nach „mit dem“ folgt im Korpus nur „ball“. Der seltsame Satz kann also nicht mehr entstehen. Der Text wird **zusammenhängender** – man sagt auch **kohärenter**.

Das hat aber einen Preis. Ein Kontext aus zwei Wörtern ist **spezifischer**: Er kommt seltener vor, und nach ihm gibt es weniger Auswahl. Im Hund-Katze-Korpus sieht das so aus:

	Bigramm-Modell	Trigramm-Modell
Kontext	1 Wort	2 Wörter
verschiedene Kontexte mit Folgewort	26	36
davon mit nur **einem** möglichen Folgewort	22	33
davon mit **Auswahl**	4	3

Beim Trigramm-Modell gibt es fast nirgends eine Auswahl. Der erzeugte Text ist dann fast immer ein Stück aus dem Korpus, Wort für Wort abgeschrieben. Er wird **weniger variantenreich**. Je länger die n-Gramme, desto stärker ist dieser Effekt. Außerdem findet ein Modell mit langen n-Grammen zu einem neuen Startkontext oft **gar keine** Fortsetzung, weil genau diese Wortfolge im Korpus nie vorkam.

Merke

Die **Länge des n-Gramms** beeinflusst die Textqualität: Größere n (z. B. Trigramme statt Bigramme) erzeugen meist **kohärentere**, aber dafür **weniger variantenreiche** Texte, da der Kontext spezifischer ist und damit seltener im Korpus vorkommt.

Arbeitsauftrag 38 (Zusatz): Einfluss der n-Gramm-Länge

Baue ein **Trigramm-Modell**, bei dem der Kontext aus einem Wortpaar besteht:

1. Schreibe eine Funktion `folgewoerter_2(woerter, wort1, wort2)`, die alle Wörter sammelt, die im Korpus direkt nach dem Wortpaar `wort1 wort2` stehen.
2. Schreibe eine Funktion `generiere_text_trigramm(woerter, wort1, wort2, laenge)`, die ab dem Startpaar mit random-Sampling einen Text erzeugt.
3. Vergleiche die erzeugten Texte mit denen des Bigramm-Modells: Wirken sie mit Trigrammen „sinnvoller“? Warum kann die Trigramm-Generierung bei einem kleinen Korpus schneller abbrechen? Probiere dazu auch das Startpaar „der ball“ aus.

Deine Antworten zu Teil 3 (die Funktionen schreibst du am Computer):

Gymnasium Ottweiler

4.4.6 Die Trainingsdaten bestimmen den Text

Ein n-Gramm-Modell kann nur Wörter und Wortfolgen erzeugen, die in seinem Korpus vorkommen. Was das bedeutet, untersuchst du im nächsten Auftrag.

Arbeitsauftrag 39: Abhängigkeit von den Trainingsdaten

Ersetze im Programm den Korpus durch einen **inhaltlich ganz anderen** Text, zum Beispiel:

```
korpus = """
heute scheint die sonne heute regnet es stark
morgen scheint die sonne morgen ist es bewölkt
heute ist es bewölkt heute regnet es leicht
"""
```

Zerlege ihn mit `split()` und erzeuge mehrmals Texte mit `generiere_text(woerter, "heute", 8, "random")`.

1. Wie stark unterscheidet sich der erzeugte Text vom Hund-Katze-Beispiel?
2. Was bedeutet das für die Aussage „Ein Sprachmodell erzeugt objektiven, neutralen Text“?
3. *Freiwillig:* Beobachte dasselbe mit SoekiaGPT, indem du dort zwischen zwei unterschiedlichen Textsammlungen wechselst und denselben Satzanfang eingibst. Das Prinzip ist identisch.

Meine Antworten:

Gymnasium Ottweiler

4.4.7 Weitere Übungen

Hier kannst du das Gelernte weiter üben. Die Punkte im Titel zeigen, wie schwer eine Aufgabe ist: ●○○ leicht, ●●○ mittel, ●●● knifflig. Fang mit den leichten an.

Mehrere Aufträge verwenden diesen Ausschnitt aus der Bigramm-Tabelle eines Korpus mit Chatnachrichten:

Kontext	mögliche Folgewörter (Anzahl)
ich	gehe (5), bin (3), habe (2)
gehe	heute (4), nach (1)
heute	ins (3), nicht (2)
ins	kino (4), bett (3)
kino	und (2)
und	ich (3), dann (1)
bin	müde (3), heute (1)
müde	und (2)

Arbeitsauftrag 40: Lückentext: next word prediction ●○○

Ergänze die Lücken. Verwende diese Wörter: *angehängt* · *endet* · *Folgewörter* · *nächste* · *Startwort* · *Tabelle*.

Bei der Textgenerierung gibt man ein _____ vor. Das Modell schaut in der n-Gramm-_____ nach, welche _____ auf das letzte Wort folgen können. Eines davon wird ausgewählt und an den Text _____. Es ist jetzt das neue letzte Wort. So wird immer wieder das _____ Wort vorhergesagt. Hat das letzte Wort keine Folgewörter, _____ der Text.

Arbeitsauftrag 41: Welche Strategie? ●○○

Ordne jeder Beschreibung die passende Sampling-Strategie zu: greedy, random oder top-k.

Nr.	Beschreibung	Strategie
1	Es wird immer das Folgewort mit der höchsten Wahrscheinlichkeit gewählt.	
2	Es wird unter allen Folgewörtern gelost. Häufige Folgewörter haben eine größere Chance.	
3	Nur die drei wahrscheinlichsten Folgewörter kommen in Frage. Unter ihnen wird gewichtet gelost.	
4	Mit demselben Startwort entsteht jedes Mal derselbe Text.	
5	Jedes Folgewort, das im Korpus vorkam, kann gewählt werden – auch das seltenste.	
6	Diese Strategie ist ein Kompromiss zwischen Vorhersagbarkeit und Abwechslung.	

Arbeitsauftrag 42: Greedy von Hand ●○○

Verwende den Tabellenausschnitt vom Anfang dieser Übungen.

1. Erzeuge mit **greedy** einen Text aus **8 Wörtern**, der mit „ich“ beginnt.
2. Beschreibe, was dir an dem Text auffällt.
3. Erkläre, warum greedy bei jedem Start genau denselben Text liefert.

Mein Text und meine Antworten:

Arbeitsauftrag 43: Wahrscheinlichkeiten bei random und top-k ●○○

In einem Korpus folgen auf „heute“ diese Wörter: „abend“ (6-mal), „nicht“ (3-mal), „schon“ (2-mal) und „mal“ (1-mal).

1. Berechne die Wahrscheinlichkeiten der Folgewörter bei **random**.
2. Berechne sie bei **top-2** und bei **top-3**.
3. Welches Wort wählt greedy?

Runde auf ganze Prozent.

Folgewort	Anzahl	random	top-2	top-3
abend	6			
nicht	3			
schon	2			
mal	1			

Arbeitsauftrag 44: Random mit Zufallszahlen ●●○

Mit random wird aus dem Tabellenausschnitt vom Anfang dieser Übungen ein Text erzeugt, der mit „ich“ beginnt. Für jeden Schritt wird eine Zufallszahl gezogen. Jedes Folgewort bekommt so viele Lose, wie es vorkommt, durchnummeriert in der Reihenfolge der Tabelle. Bei „ich“ heißt das: Lose 1 bis 5 sind „gehe“, 6 bis 8 „bin“, 9 und 10 „habe“.

Die Zufallszahlen sind der Reihe nach: **7, 4, 2, 1, 2, 4**.

1. Bestimme den erzeugten Text. Notiere bei jedem Schritt, welche Lose zu welchem Wort gehören.
2. Kann greedy diesen Text erzeugen? Begründe.
3. Mit welcher Wahrscheinlichkeit wählt random nach „ich“ das Wort „bin“?

Meine Schritte und Antworten:

Arbeitsauftrag 45: Welcher Text ist möglich? ●●○

Ein Bigramm-Modell wurde mit diesem Korpus trainiert:

> lisa fährt mit dem bus tim fährt mit dem rad lisa geht zu fuß

Entscheide für jeden Text, ob das Modell ihn mit **random** erzeugen kann. Wenn nicht, nenne ein Wortpaar im Text, das im Korpus nicht vorkommt.

Text	möglich?	fehlendes Wortpaar
tim fährt mit dem bus		
lisa fährt mit dem auto		
dem rad lisa geht zu fuß		
lisa geht mit dem bus		

Welcher der möglichen Texte kommt so **nicht** im Korpus vor?

Antwort zur letzten Frage:

Arbeitsauftrag 46: Bigramme oder Trigramme? ●●○

Gegeben ist der Korpus:

> ich trinke einen kakao ich esse einen apfel ich trinke einen tee

1. Bestimme im **Bigramm-Modell** die Folgewörter von „einen“ mit ihren Wahrscheinlichkeiten.
2. Bestimme im **Trigramm-Modell** die Folgewörter der Kontexte „trinke einen“ und „esse einen“ mit ihren Wahrscheinlichkeiten.
3. Kann das Bigramm-Modell den Text „ich trinke einen apfel“ erzeugen? Und das Trigramm-Modell?
4. Erkläre an diesem Beispiel, warum größere n kohärentere, aber weniger variantenreiche Texte erzeugen.

Meine Antworten:

Gymnasium Ottweiler

Arbeitsauftrag 47: Die Trainingsdaten entscheiden ●●●

Zwei Bigramm-Modelle wurden mit verschiedenen Korpora trainiert:

- **Verein:** „wir trainieren hart wir trainieren lange wir gewinnen das spiel wir trainieren hart“
- **Schule:** „wir lernen vokabeln wir lernen mathe wir schreiben eine arbeit wir lernen vokabeln“

1. Bestimme für beide Modelle den Text, den **greedy** ab „wir“ mit **5 Wörtern** erzeugt.
2. Erkläre, warum dasselbe Verfahren mit demselben Startwort ganz verschiedene Texte erzeugt.
3. An den Schul-Korpus wird mehrmals der Satz „wir schreiben eine arbeit“ angehängt. Wie oft mindestens, damit greedy nach „wir“ das Wort „schreiben“ wählt? Bei Gleichstand wählt greedy das Folgewort, das im Korpus zuerst vorkommt.
4. Prüfe deine Antworten mit der Funktion `generiere_text` aus diesem Abschnitt.

Meine Antworten zu 1 bis 3:

4.4.8 Das Wichtigste in Kürze

- Bei der **Textgenerierung (next word prediction)** wird ab einem Startwort immer wieder das nächste Wort bestimmt und angehängt. Hat ein Wort keine Folgewörter, endet der Text.
- Die **Sampling-Strategie** legt fest, wie das Folgewort ausgewählt wird:
  - **greedy:** immer das wahrscheinlichste → deterministisch, aber eintönig, oft Wiederholungen,
  - **random:** gewichtet zufällig (Lostrommel) → abwechslungsreich, manchmal unsinnig,
  - **top-k:** gewichtet zufällig unter den k wahrscheinlichsten → Kompromiss.
- Bei top-k werden die Wahrscheinlichkeiten der übrigen Wörter neu berechnet (z. B. $\frac{4}{7}$ statt $\frac{4}{10}$).
- **Längere n-Gramme** liefern kohärentere, aber weniger variantenreiche Texte. Lange Kontexte kommen im Korpus seltener vor.
- Der erzeugte Text hängt vollständig von den **Trainingsdaten** ab. Ein Sprachmodell ist deshalb nicht automatisch neutral.

Selbstcheck

- ☐ Ich kann beschreiben, wie ein n-Gramm-Modell Wort für Wort einen Text erzeugt.
- ☐ Ich kann die Strategien greedy, random und top-k benennen und ihre Vor- und Nachteile nennen.
- ☐ Ich kann mit Zufallszahlen von Hand ein Folgewort nach random oder top-k auslosen.
- ☐ Ich kann die Wahrscheinlichkeiten bei top-k neu berechnen und als Balkendiagramm darstellen.
- ☐ Ich kann erklären, warum die Liste `folgewoerter(...)` eine Lostrommel für gewichtetes Ziehen ist.
- ☐ Ich kann beschreiben, wie sich längere n-Gramme auf den erzeugten Text auswirken.
- ☐ Ich kann erläutern, warum der erzeugte Text von den Trainingsdaten abhängt.

4.5 Halluzinationen, moderne KI und die Arbeitswelt

Leitidee

Ein Sprachmodell schreibt einen Satz, der flüssig klingt und grammatisch einwandfrei ist – und trotzdem ist er falsch. Fachleute nennen das **Halluzination**. Das passiert nicht nur kleinen Modellen, sondern auch großen Chatbots. Mit deinem Wissen über n-Gramm-Modelle kannst du jetzt genau erklären, **warum** das passiert. Danach vergleichst du das einfache n-Gramm-Modell mit modernen Sprachmodellen und überlegst, was das für deinen Umgang mit KI bedeutet. Zum Schluss blickst du in die Arbeitswelt: Wie verändert generative KI Berufe, und welche neuen entstehen?

Lernziele

Nach diesem Abschnitt kannst du

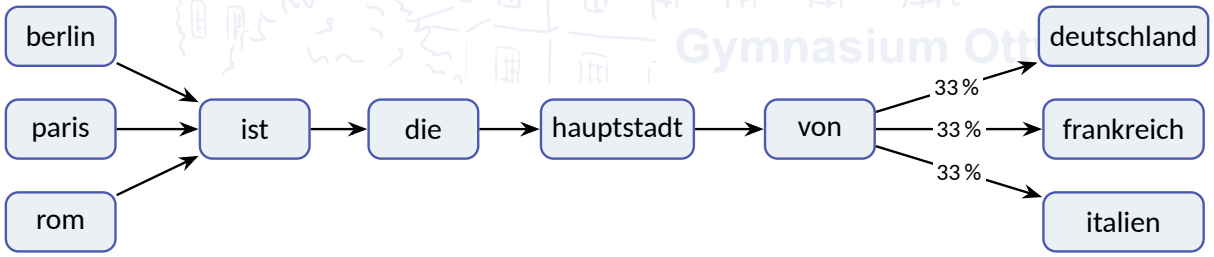
- erklären, was eine Halluzination ist und wie sie bei n-Gramm-Modellen entsteht,
- an einem Beispiel zeigen, warum ein flüssig klingender Satz inhaltlich falsch sein kann,
- Unterschiede zwischen n-Gramm-Modellen und modernen LLMs in Sprachqualität, Korrektheit, Kontextberücksichtigung und Vielseitigkeit beschreiben,
- begründen, warum man Aussagen von KI-Systemen selbst überprüfen muss,
- Auswirkungen generativer KI auf ein Berufsfeld recherchieren und darstellen.

4.5.1 Ein Satz, der nur richtig klingt

Wir trainieren ein Bigramm-Modell mit einem sehr kleinen Korpus aus drei richtigen Sätzen:

```
berlin ist die hauptstadt von deutschland
paris ist die hauptstadt von frankreich
rom ist die hauptstadt von italien
```

Die Bigramm-Tabelle ist schnell aufgestellt. Auf „berlin“, „paris“ und „rom“ folgt jeweils „ist“, darauf „die“, dann „hauptstadt“ und dann „von“. Erst nach „von“ gibt es eine Auswahl:



Bigramm-Modell zum Hauptstadt-Korpus: Erst nach „von“ gibt es drei mögliche Folgewörter. Welche Stadt am Anfang stand, spielt dafür keine Rolle

Jetzt lassen wir das Modell mit dem Startwort „paris“ einen Satz aus sechs Wörtern schreiben, und zwar mit **greedy**. Bis „von“ gibt es keine Wahl: „paris ist die hauptstadt von“. Nach „von“ haben alle drei Länder dieselbe Wahrscheinlichkeit. Greedy nimmt bei Gleichstand das Wort, das im Korpus zuerst vorkommt, also „deutschland“. Das Ergebnis:

> paris ist die hauptstadt von deutschland

Der Satz ist grammatisch perfekt. Jedes Wortpaar darin kommt genau so im Trainingstext vor. Und trotzdem ist die Aussage **falsch**. Mit random wäre es nicht besser: Dann ist der Satz in zwei von drei Fällen falsch, egal mit welcher Stadt er beginnt.

Woran liegt das? Wenn das Modell das Wort nach „von“ auswählt, schaut es **nur auf „von“**. Dass am Anfang „paris“ stand, hat es längst „vergessen“ – es hat es nie beachtet. Das Modell weiß auch nicht, was eine Hauptstadt ist oder was die Wörter bedeuten. Es reiht Wörter nach Wahrscheinlichkeit aneinander. Ob der Satz wahr ist, spielt dabei keine Rolle.

Definition: Halluzination

Eine **Halluzination** ist ein von einem Sprachmodell erzeugter Text, der sprachlich plausibel klingt, aber inhaltlich falsch oder unsinnig ist.

Merke

- **Halluzination** bezeichnet das Phänomen, dass ein Sprachmodell Text erzeugt, der **grammatikalisch bzw. statistisch plausibel**, aber **inhaltlich falsch oder unsinnig** ist.
- Bei n-Gramm-Modellen entsteht Halluzination, weil das Modell **keinerlei „Verständnis“** der Bedeutung hat. Es reiht Wörter nur nach **statistischer Wahrscheinlichkeit** aneinander, unabhängig davon, ob der entstehende Satz sachlich korrekt ist.

Das Wort *Halluzination* kommt eigentlich aus der Medizin. Dort bezeichnet es Wahrnehmungen von Dingen, die gar nicht da sind. Bei Sprachmodellen ist der Begriff ein Bild: Das Modell „erfindet“ etwas, ohne es zu merken. In Wirklichkeit merkt es gar nichts – es rechnet nur.

So kannst du es sagen

So erklärst du eine Halluzination in Fachsprache:

- *„Das Modell hat sich vertan.“ → „Das Modell hat eine Halluzination erzeugt: Der Satz ist grammatisch korrekt, aber inhaltlich falsch.“*
- *„Es hat Paris vergessen.“ → „Das Bigramm-Modell berücksichtigt nur das letzte Wort als Kontext. Deshalb wählt es das Folgewort von ‚von‘ unabhängig vom Satzanfang aus.“*

Arbeitsauftrag 48: Halluzinationen im Hauptstadt-Modell

Verwende das Bigramm-Modell zum Hauptstadt-Korpus.

1. Das Modell erzeugt mit **random** einen Satz aus sechs Wörtern, der mit „rom“ beginnt. Wie groß ist die Wahrscheinlichkeit, dass der Satz inhaltlich richtig ist? Wie groß ist sie, dass er falsch ist?
2. Würde ein **Trigramm-Modell** (Kontext aus zwei Wörtern) das Problem lösen? Begründe.
3. Wie lang müsste der Kontext mindestens sein, damit das Modell immer das richtige Land wählt? Was ist dann der Nachteil?

Meine Antworten:

4.5.2 Halluzinationen im Hund-Katze-Modell

Auch mit dem Hund-Katze-Korpus aus den letzten Abschnitten entstehen leicht seltsame Sätze. Das folgende Programm erzeugt mit random fünf Texte, die mit „die“ beginnen. Es enthält nur die Funktionen, die dafür nötig sind. Die Funktion `generiere_text_random` ist eine kürzere Fassung von `generiere_text` aus dem Abschnitt über Textgenerierung, die immer mit random auswählt.

```python
import random

def folgewoerter(woerter, wort):
    folge = []
    for i in range(len(woerter) - 1):
        if woerter[i] == wort:
            folge.append(woerter[i + 1])
    return folge

def generiere_text_random(woerter, startwort, laenge):
    text = startwort
    aktuelles_wort = startwort
    for schritt in range(laenge - 1):
        folge = folgewoerter(woerter, aktuelles_wort)
        if len(folge) == 0:
            break
        aktuelles_wort = folge[random.randint(0, len(folge) - 1)]
        text = text + " " + aktuelles_wort
    return text

korpus = """
der hund läuft im park der hund spielt mit dem ball
die katze sitzt auf dem baum die katze beobachtet den hund
der hund bellt laut der hund läuft zur katze
die katze springt vom baum die katze rennt weg
der hund läuft ihr hinterher der hund ist müde
"""
woerter = korpus.split()
for nummer in range(5):
    print(generiere_text_random(woerter, "die", 8))
```

AUSGABE

```
die katze sitzt auf dem ball die katze
die katze beobachtet den hund läuft zur katze
die katze springt vom baum die katze rennt
die katze rennt weg der hund ist müde
die katze die katze sitzt auf dem baum
```

Die Ausgabe ist ein **Beispiel**. Bei jedem Start entstehen andere Texte.

- **Zeilen 3–8:** `folgewoerter` sammelt alle Folgewörter eines Wortes, mit Wiederholungen. Diese Liste ist die Lostrommel.
- **Zeilen 10–19:** In jedem Schritt wird ein Los aus der Lostrommel des aktuellen Wortes gezogen und angehängt. Ohne Folgewörter endet der Text.
- **Zeilen 29–30:** Fünf Texte mit je höchstens acht Wörtern werden erzeugt.

Schau dir die erste Zeile an: „die katze sitzt auf dem ball“. Im Korpus sitzt die Katze auf dem **Baum**, und mit dem **Ball** spielt der Hund. Das Modell hat nach „dem“ zwischen „ball“ und „baum“ gelost – beide Wortpaare sind im Korpus richtig, der Satz insgesamt ist eine Erfindung. Auch „den hund läuft zur katze“ ist grammatisch holprig, weil nach „hund“ nicht beachtet wird, dass davor „den“ stand.

Arbeitsauftrag 49: Halluzinationen einordnen

Führe das Programm mehrmals aus. Suche einen erzeugten Satz heraus, der **grammatikalisch plausibel, aber inhaltlich seltsam oder unsinnig** ist. Erkläre in eigenen Worten, **warum** genau dieser Fehler entsteht. Beziehe dich dabei auf die Funktionsweise des n-Gramm-Modells.

Mein Beispielsatz und meine Erklärung:

Versuch: Probiere es aus: Halluzinationen in SoekiaGPT

Wenn deine SoekiaGPT-Version eigene Texte erlaubt, lege eine eigene kleine Textsammlung an, zum Beispiel mit drei oder vier kurzen Steckbriefen von Fantasietieren, die ähnlich aufgebaut sind („der glitzerfuchs lebt im wald und frisst beeren …“). Lass dann Texte erzeugen, die mit dem Namen eines Tieres beginnen. Werden Eigenschaften verschiedener Tiere vermischt? Schau dir in der n-Gramm-Anzeige auch längere n-Gramme an: Welche Wortfolgen stammen aus nur einem einzigen Steckbrief?

4.5.3 n-Gramm-Modelle und moderne LLMs

Moderne Chatbots beruhen auf **Large Language Models (LLMs)**. Sie schätzen ebenfalls Wahrscheinlichkeiten für die Fortsetzung eines Textes, sind aber in vieler Hinsicht viel leistungsfähiger als unser n-Gramm-Modell. Ein LLM wurde mit unvorstellbar viel mehr Text trainiert und betrachtet nicht nur ein oder zwei Wörter, sondern sehr lange Textabschnitte als Kontext. Dadurch „vergisst“ es den Satzanfang nicht so schnell wie unser Modell.

Trotzdem bleibt das Grundprinzip gleich: Ein LLM erzeugt Text, **der wahrscheinlich klingt**. Es schlägt nicht in einer geprüften Faktensammlung nach. Wenn eine Information in den Trainingsdaten selten, widersprüchlich oder gar nicht vorkommt, erzeugt es trotzdem eine flüssige Antwort. Bekannte Beispiele sind erfundene Quellenangaben mit Büchern oder Artikeln, die es nicht gibt, falsche Jahreszahlen oder erfundene Zitate.

Merke

Moderne **LLMs** unterscheiden sich von einfachen n-Gramm-Modellen u. a. in:

- **Sprachqualität:** LLMs erzeugen deutlich flüssigeren, grammatikalisch korrekteren Text.
- **Kontextberücksichtigung:** LLMs können sehr viel längere Kontexte einbeziehen (nicht nur 1–2 vorherige Wörter).

- **Korrektheit:** LLMs erzeugen häufig zuverlässigere Antworten, **können aber weiterhin halluzinieren**. Das n-Gramm-Modell dient hierfür als **vereinfachte Analogie**: Auch LLMs erzeugen Text Wort für Wort nach Wahrscheinlichkeiten und prüfen ihn nicht an Fakten. Die Ursachen bei modernen LLMs sind aber wesentlich komplexer und nicht mit dem einfachen Modell gleichzusetzen.
- **Vielseitigkeit:** LLMs können unterschiedlichste Aufgaben bewältigen (Übersetzen, Zusammenfassen, Programmieren …), während ein einfaches n-Gramm-Modell nur Text im Stil seines Trainingskorpus fortsetzt.

Arbeitsauftrag 50: Vergleichstabelle

Stelle n-Gramm-Modelle und moderne LLMs in einer Tabelle gegenüber. Beschreibe für jedes Kriterium in eigenen Worten, was du beobachtet hast, und nenne – wo möglich – ein Beispiel aus diesem Kapitel.

Kriterium	n-Gramm-Modell	modernes LLM
Sprachqualität		
Korrektheit		
Kontextberücksichtigung		
Vielseitigkeit		

4.5.4 Fakten selbst prüfen

Weil auch große Sprachmodelle halluzinieren können, gilt: **Was ein KI-System schreibt, ist ein Vorschlag, keine geprüfte Tatsache.** Gerade weil die Texte so überzeugend klingen, fallen Fehler leicht nicht auf. Das ist auch für unsere Gesellschaft wichtig: Falschinformationen (**Desinformation**) verbreiten sich schnell, und wer Aussagen nicht prüft, kann leichter getäuscht oder manipuliert werden. In einer Demokratie ist es wichtig, dass sich Menschen ihre Meinung auf Grundlage zutreffender Informationen bilden.

So gehst du vor, wenn du eine Aussage eines KI-Systems verwenden willst:

- **Unabhängig prüfen:** Suche die Aussage in mindestens einer seriösen, unabhängigen Quelle (z. B. Lehrbuch, Fachlexikon, Website einer Behörde oder Hochschule, Qualitätszeitung).
- **Quellen nachschlagen:** Nennt das System Quellen, prüfe, ob es sie wirklich gibt und ob dort tatsächlich das steht, was behauptet wird.
- **Besonders vorsichtig sein** bei Zahlen, Jahreszahlen, Namen, Zitaten und Literaturangaben. Hier entstehen Halluzinationen besonders leicht.
- **Aktualität beachten:** Ein Sprachmodell selbst kennt nur, was in seinen Trainingsdaten stand. Über neuere Ereignisse weiß ein Chatbot nur etwas, wenn er zusätzlich im Internet sucht – und auch dann können Fehler entstehen.

Merke

KI-Systeme erzeugen Texte nach Wahrscheinlichkeit, nicht nach Wahrheit. Wichtige Aussagen muss man deshalb **selbst mit seriösen Quellen überprüfen** – das schützt vor Irrtümern, Desinformation und Manipulation.

Arbeitsauftrag 51: Stellung nehmen

Deine Mitschülerin sagt: „Was der Chatbot schreibt, stimmt bestimmt. Er klingt doch total sicher und hat sogar Quellen angegeben.“

Nimm Stellung zu dieser Aussage. Verwende dabei, was du über Halluzinationen weißt.

Meine Stellungnahme:

4.5.5 Berufliche Orientierung: Generative KI in der Arbeitswelt

Sprachmodelle und andere **generative KI-Systeme** (Systeme, die Texte, Bilder, Musik oder Programmcode erzeugen) werden inzwischen in vielen Berufen eingesetzt. Manche Tätigkeiten werden dadurch schneller oder einfacher, manche fallen weg, und es entstehen neue Aufgaben – zum Beispiel das geschickte Formulieren von Anweisungen an eine KI, das **Prompt Engineering**, oder das Prüfen und Überarbeiten von KI-Ergebnissen.

Wie stark ein Beruf betroffen ist, lässt sich nicht pauschal sagen. Deshalb recherchierst du selbst, am besten zu zweit oder zu dritt.

Arbeitsauftrag 52: Generative KI und Berufe

Recherchiert die Auswirkungen generativer KI-Systeme auf **ein** Berufsfeld eurer Wahl, zum Beispiel:

- Journalismus und Erstellen von Inhalten für Websites und soziale Medien,
- Softwareentwicklung,
- Grafikdesign und Illustration,
- Kundenservice und Support,
- Übersetzen und Dolmetschen.

Leitfragen für eure Recherche:

1. Welche Tätigkeiten dieses Berufsfeldes werden bereits heute durch generative KI unterstützt oder verändert?

2. Entstehen durch generative KI **neue** Aufgaben oder sogar neue Berufsbilder in diesem Bereich (z. B. Prompt Engineering)?
3. Welche Tätigkeiten lassen sich aus heutiger Sicht **nicht** oder nur schwer durch KI ersetzen – und warum?
4. Wie bewertet ihr diese Entwicklung? Welche Chancen und welche Risiken seht ihr?
5. Findet (falls möglich) eine **reale Person** mit Bezug zu diesem Berufsfeld und recherchiert ihre Berufsbiografie (Ausbildung, erste Stelle, heutige Tätigkeit), z. B. über ein Interview, eine Unternehmenswebsite oder ein Karriereportal.

Haltet eure Ergebnisse in Stichpunkten in der Ergebnisvorlage unten fest und bereitet einen kurzen mündlichen Beitrag von etwa einer Minute vor.

Tipp

Seriöse Quellen für die Recherche sind zum Beispiel: das Berufsinformationsportal **BERUFENET** und die Seite **planet-beruf.de** der Bundesagentur für Arbeit, das Berufsinformationszentrum (BIZ) der Agentur für Arbeit, Berufsverbände und Kammern (z. B. IHK, Handwerkskammer), Hochschulen sowie Berichte und Interviews in Qualitätsmedien.

Achte bei jeder Quelle darauf, **wer** sie veröffentlicht hat und **wann**. Die Entwicklung ist schnell, ältere Texte können überholt sein. Übernimm Zahlen (z. B. „Anteil der Jobs, die …“) nur, wenn du die Quelle nennen kannst, und notiere, worauf sich die Zahl bezieht. Wenn du einen Chatbot zur Recherche nutzt, prüfe seine Aussagen wie im Abschnitt *Fakten selbst prüfen* beschrieben.

Ergebnisvorlage: Leitfrage	**unsere Ergebnisse (Stichpunkte)**
Berufsfeld	
1. bereits veränderte oder unterstützte Tätigkeiten	
2. neue Aufgaben und Berufsbilder	
3. (noch) nicht ersetzbare Tätigkeiten und warum	
4. Chancen	
4. Risiken	
5. Berufsbiografie einer realen Person	
Quellen (Herausgeber, Titel, Datum)	

Notizen für den mündlichen Beitrag:

4.5.6 Weitere Übungen

Hier kannst du das Gelernte weiter üben. Die Punkte im Titel zeigen, wie schwer eine Aufgabe ist: ●○○ leicht, ●●○ mittel, ●●● knifflig. Fang mit den leichten an.

Mehrere Aufträge verwenden ein Bigramm-Modell, das mit diesem Korpus aus zwei Steckbriefen trainiert wurde:

> lina wohnt in ottweiler und spielt handball tom wohnt in neunkirchen und spielt schach

Arbeitsauftrag 53: Halluzination oder nicht? ●○○

Das Modell hat die folgenden Sätze erzeugt. Entscheide jeweils, ob der Satz zu den Steckbriefen im Korpus passt oder ob er eine **Halluzination** ist.

Nr.	erzeugter Satz	passt / Halluzination
1	lina wohnt in ottweiler und spielt handball	
2	tom wohnt in ottweiler und spielt schach	
3	tom wohnt in neunkirchen und spielt schach	
4	lina wohnt in neunkirchen und spielt handball	
5	tom wohnt in neunkirchen und spielt handball	

Arbeitsauftrag 54: Richtig oder falsch? ●○○

Entscheide, ob die Aussage richtig oder falsch ist. Trage „r“ oder „f“ ein. Verbessere die falschen Aussagen unten.

Nr.	Aussage	r oder f?
1	Eine Halluzination ist ein Text, der plausibel klingt, aber inhaltlich falsch oder unsinnig ist.	
2	Ein n-Gramm-Modell halluziniert, weil es absichtlich falsche Dinge schreibt.	
3	Moderne LLMs können nicht mehr halluzinieren.	
4	LLMs können viel längere Kontexte einbeziehen als n-Gramm-Modelle.	
5	Ein einfaches n-Gramm-Modell setzt nur Text im Stil seines Trainingskorpus fort.	
6	Mit dem n-Gramm-Modell sind die Ursachen von Halluzinationen bei LLMs vollständig erklärt.	

Verbesserungen:

Arbeitsauftrag 55: Kriterien zuordnen ●○○

Ordne jede Beobachtung einem der vier Kriterien zu: **Sprachqualität**, **Kontextberücksichtigung**, **Korrektheit** oder **Vielseitigkeit**.

Nr.	Beobachtung	Kriterium
1	Ein LLM übersetzt einen Text ins Englische und fasst ihn danach zusammen.	
2	Das Bigramm-Modell achtet bei der Wahl des nächsten Wortes nur auf das letzte Wort.	
3	Ein LLM schreibt flüssige, grammatisch richtige Sätze.	
4	Ein LLM nennt eine falsche Jahreszahl, obwohl die Antwort sehr sicher klingt.	
5	Das n-Gramm-Modell erzeugt „den hund läuft zur katze“.	
6	Ein LLM bezieht sich in seiner Antwort auf eine Frage, die viel weiter oben im Gespräch stand.	

Arbeitsauftrag 56: Halluzinationen erklären ●●○

Verwende das Bigramm-Modell zu den Steckbriefen.

1. Bestimme die Folgewörter von „in“ und von „spielt“ mit ihren Wahrscheinlichkeiten.
2. Das Modell erzeugt mit **random** einen Text aus **7 Wörtern**, der mit „tom“ beginnt. Nenne alle Texte, die dabei entstehen können.
3. Wie viele dieser Texte passen zum Korpus, wie viele sind Halluzinationen?
4. Erkläre mit der Funktionsweise des Bigramm-Modells, warum die Halluzinationen entstehen.

Meine Antworten:

Arbeitsauftrag 57: Kundenservice und KI ●●○

Im Kundenservice eines Online-Shops fallen diese Tätigkeiten an:

a) häufige Standardfragen im Chat beantworten (z. B. „Wo ist mein Paket?“),
b) Antworten auf E-Mails vorformulieren,
c) eine sehr verärgerte Kundin am Telefon beruhigen und eine faire Lösung mit ihr finden,
d) eingehende Anfragen nach Themen sortieren,
e) entscheiden, ob ein Kunde in einem Sonderfall sein Geld zurückbekommt.

1. Ordne jede Tätigkeit zu: „wird schon heute durch generative KI unterstützt“ oder „aus heutiger Sicht schwer zu ersetzen“. Begründe die zweite Gruppe.
2. Nenne eine **neue** Aufgabe, die in diesem Berufsfeld durch generative KI entsteht.

Meine Zuordnung und Begründung:

Arbeitsauftrag 58: Hilft ein längerer Kontext? ●●●

Verwende wieder den Korpus mit den Steckbriefen.

1. Ein **Trigramm-Modell** (Kontext aus zwei Wörtern) erzeugt einen Text, der mit „tom wohnt“ beginnt. Kann dabei noch „tom wohnt in ottweiler“ entstehen? Begründe.
2. Bestimme, wie viele Wörter der Kontext **mindestens** umfassen muss, damit das Modell ab „tom wohnt in“ nur noch den richtigen Steckbrief erzeugen kann. Begründe für den Wohnort **und** für das Hobby.
3. Beschreibe, welchen Nachteil ein so langer Kontext hat.

Gymnasium Ottweiler

Meine Antworten:

4.5.7 Das Wichtigste in Kürze

- Eine **Halluzination** ist ein Text, der sprachlich plausibel klingt, aber inhaltlich falsch oder unsinnig ist.
- Beim n-Gramm-Modell entsteht sie, weil nur die letzten ein bis zwei Wörter als Kontext zählen und die Bedeutung keine Rolle spielt: Jedes Wortpaar stammt aus dem Korpus, die Kombination kann trotzdem falsch sein. Längere n-Gramme helfen nur, bis das Modell bloß noch abschreibt.
- **LLMs** sind besser in Sprachqualität, Kontextberücksichtigung und Vielseitigkeit und meist zuverlässiger – sie halluzinieren aber aus demselben Grund.
- Aussagen von KI-Systemen muss man **selbst mit seriösen Quellen prüfen**. Das schützt vor Desinformation und Manipulation.
- Generative KI verändert Berufe: Tätigkeiten werden unterstützt oder fallen weg, neue Aufgaben entstehen.

Selbstcheck

- ☐ Ich kann erklären, was eine Halluzination ist.
- ☐ Ich kann an einem n-Gramm-Beispiel zeigen, warum ein flüssiger Satz falsch sein kann.
- ☐ Ich kann begründen, warum ein längerer Kontext hilft, aber auch Nachteile hat.
- ☐ Ich kann n-Gramm-Modelle und LLMs in vier Kriterien vergleichen.
- ☐ Ich kann beschreiben, wie ich eine Aussage eines KI-Systems überprüfe.

- ☐ Ich kann zu einem Berufsfeld recherchieren, wie generative KI es verändert, und meine Quellen angeben.

4.6 Kapitel 4 kompakt: der Lernzettel

Leitidee

Hier steht das Wichtigste aus Kapitel 4 auf wenigen Seiten, zum Wiederholen vor einem GLN. Wenn dir etwas unklar ist, lies im passenden Abschnitt nach: Dort ist alles ausführlich erklärt, mit Beispielen und Aufgaben.

4.6.1 Sprachmodelle: Regeln oder Statistik? (Abschnitt 4.1)

- Ein **Sprachmodell** ist ein System, das sprachliche Strukturen bzw. Wahrscheinlichkeiten erfasst, um Texte zu analysieren oder neu zu erzeugen.
- **Regelbasierte Systeme** (z. B. **ELIZA**, 1966) reagieren nach **fest programmierten Regeln** und Mustererkennung: „Wenn der Satz ‚ich fühle mich …' enthält, antworte ‚Warum fühlst du dich …?' ". Die Regeln werden von oben nach unten geprüft, die erste passende gewinnt.
- **n-Gramm-Modelle** zählen in großen Textmengen, wie häufig Wortfolgen vorkommen, und sagen damit das **nächste Wort** voraus.
- Moderne **LLMs** (Large Language Models) nutzen komplexe neuronale Netze, schätzen im Kern aber ebenfalls **Wahrscheinlichkeiten für die Fortsetzung eines Textes**.

	regelbasiertes System	**n-Gramm-Modell**
Woher kommt das Wissen?	Menschen schreiben Regeln.	Der Computer zählt in Texten.
Grenze	Für jede Formulierung bräuchte man eine Regel.	Kann nur, was im Korpus vorkam.
Versteht es die Bedeutung?	nein	nein

Typische Fehler: Meinen, ELIZA „verstehe" den Satz. Vergessen, dass die Reihenfolge der Regeln entscheidet, wenn mehrere passen.

4.6.2 n-Gramme zählen, Wahrscheinlichkeiten berechnen (Abschnitte 4.2 und 4.3)

- Ein **n-Gramm** ist eine Folge von n aufeinanderfolgenden Wörtern (oder Zeichen). Die Stücke überlappen sich.

Name	**n**	**aus „der hund bellt laut"**
Unigramm	1	der · hund · bellt · laut
Bigramm	2	der hund · hund bellt · bellt laut
Trigramm	3	der hund bellt · hund bellt laut

- Der **Textkorpus** ist der Trainingstext. Bei der **Textkörperanalyse** zählt man, wie oft jedes n-Gramm darin vorkommt.
- Ein Text mit 9 Wörtern hat 8 Bigramme: Das letzte Wort hat keinen Nachfolger.
- Der **Kontext** ist das, worauf das Modell schaut: beim Bigramm ein Wort, beim Trigramm zwei Wörter.
- **Wahrscheinlichkeit** eines Folgewortes:

$$\text{Wahrscheinlichkeit} = \frac{\text{Anzahl des n-Gramms „Kontext Folgewort"}}{\text{Anzahl aller n-Gramme, die mit dem Kontext beginnen}}$$

- Kontrolle: Für jeden Kontext ergeben die Wahrscheinlichkeiten zusammen 1 (100 %).

Beispiel zum Korpus „ich mag tee ich mag kaffee ich trinke tee“ (9 Wörter, 8 Bigramme):

Kontext	Folgewort	Anzahl	Wahrscheinlichkeit
ich	mag	2	$\frac{2}{3} \approx 67\,\%$
ich	trinke	1	$\frac{1}{3} \approx 33\,\%$
mag	tee	1	$\frac{1}{2} = 50\,\%$
mag	kaffee	1	$\frac{1}{2} = 50\,\%$
tee	ich	1	$\frac{1}{1} = 100\,\%$
kaffee	ich	1	$\frac{1}{1} = 100\,\%$
trinke	tee	1	$\frac{1}{1} = 100\,\%$

- Große Korpora zählt man **automatisiert** mit Software (z. B. SoekiaGPT oder ein eigenes Python-Programm).
- Die Ergebnisse werden **tabellarisch** verwaltet: zu jedem Kontext die Folgewörter mit Anzahl. Gängige Datenstruktur: ein **Dictionary von Dictionaries** `{kontext: {folgewort: anzahl}}`, für das Beispiel oben:

```
{"ich": {"mag": 2, "trinke": 1}, "mag": {"tee": 1, "kaffee": 1}, ...}
```

Alle Bigramme eines Textes ausgeben:

```
woerter = korpus.split()                # Text -> Liste von Wörtern
for i in range(len(woerter) - 1):       # bis zum vorletzten Wort!
    print(woerter[i], woerter[i + 1])
```

Typische Fehler: Durch die Anzahl **aller** Bigramme teilen statt durch die Bigramme, die mit dem Kontext beginnen. Das **letzte Wort** mitzählen, obwohl ihm kein Wort folgt. `range(len(woerter))` statt `range(len(woerter) - 1)` führt zum `IndexError`. „Der“ und „der“ sind für den Computer verschiedene Wörter, deshalb alles kleinschreiben.

4.6.3 Texte erzeugen: next word prediction und Sampling (Abschnitt 4.4)

- **Textgenerierung (next word prediction):** Ab einem Startwort wird mithilfe der n-Gramm-Tabelle immer wieder das **nächste Wort** bestimmt und angehängt. Hat das letzte Wort keine Folgewörter, endet der Text.
- Die **Sampling-Strategie** legt fest, **wie** aus mehreren Folgewörtern eines ausgewählt wird:

Strategie	Auswahl	Folge
greedy	immer das **wahrscheinlichste** Folgewort	deterministisch, oft eintönig und sich wiederholend
random	**zufällig, gewichtet** nach Wahrscheinlichkeit	abwechslungsreicher, manchmal unsinnig
top-k	nur die k wahrscheinlichsten, darunter gewichtet zufällig	Kompromiss zwischen Vorhersagbarkeit und Abwechslung

- **Top-k neu rechnen:** Folgewörter 4-mal, 3-mal, 2-mal, 1-mal. Bei random $\frac{4}{10}, \frac{3}{10}, \frac{2}{10}, \frac{1}{10}$. Bei top-2 bleiben zwei Wörter mit $4 + 3 = 7$ Vorkommen: $\frac{4}{7}$ und $\frac{3}{7}$.

- **Länge des n-Gramms:** Größere n (z. B. Trigramme statt Bigramme) ergeben **kohärentere**, aber **weniger variantenreiche** Texte. Der längere Kontext ist spezifischer und kommt seltener im Korpus vor. Bei kleinem Korpus bricht die Generierung schneller ab.
- **Trainingsdaten:** Das Modell kann nur Wörter und Wortfolgen aus seinem Korpus erzeugen. Ein anderer Korpus ergibt ganz andere Texte. Ein Sprachmodell erzeugt deshalb **nicht automatisch neutralen** Text.

Typische Fehler: Bei top-k weiter durch die alte Gesamtzahl teilen. Glauben, random wähle alle Wörter gleich oft: Häufige Wörter werden öfter gezogen. Vergessen, dass greedy immer denselben Text liefert.

4.6.4 Halluzination und moderne LLMs (Abschnitt 4.5)

- Eine **Halluzination** ist ein Text, der **grammatikalisch bzw. statistisch plausibel**, aber **inhaltlich falsch oder unsinnig** ist.
- Ursache beim n-Gramm-Modell: Es hat **kein Verständnis** der Bedeutung. Es reiht Wörter nach **statistischer Wahrscheinlichkeit** aneinander und schaut nur auf den kurzen Kontext. Jedes Wortpaar stammt aus dem Korpus, die Kombination kann trotzdem falsch sein.

Kriterium	n-Gramm-Modell	modernes LLM
Sprachqualität	oft holprig	flüssig, grammatisch meist korrekt
Kontextberücksichtigung	nur 1–2 vorherige Wörter	sehr lange Kontexte
Korrektheit	Inhalt oft zufällig falsch	häufig zuverlässiger, kann aber **weiterhin halluzinieren**
Vielseitigkeit	setzt nur Text im Stil seines Korpus fort	übersetzen, zusammenfassen, programmieren …

- Das n-Gramm-Modell ist für Halluzinationen bei LLMs nur eine **vereinfachte Analogie**. Die Ursachen bei LLMs sind wesentlich komplexer.
- **Generative KI und Berufe:** Manche Tätigkeiten werden unterstützt oder verändert, es entstehen neue Aufgaben (z. B. Prompt Engineering), manche Tätigkeiten sind schwer zu ersetzen. Zu einer Bewertung gehören Chancen **und** Risiken.

Typische Fehler: „Das Modell lügt“ – es hat keine Absicht, es rechnet nur. „LLMs halluzinieren nicht mehr“ – doch, auch sie können es. Eine Halluzination nur „falsch“ nennen, ohne die Ursache (Wahrscheinlichkeit statt Bedeutung, kurzer Kontext) zu erklären.

4.7 Rückblick auf Kapitel 4

Leitidee

Am Ende des Kapitels lohnt sich ein Blick zurück: Welche Fachbegriffe kennst du jetzt? Was kannst du schon sicher, was solltest du noch einmal üben?

4.7.1 Die Basisbegriffe

Die ersten sechs Begriffe nennt der Lehrplan als **Basisbegriffe**. Du solltest jeden davon mit eigenen Worten erklären und an einem kleinen Beispiel zeigen können.

Begriff	Bedeutung
Sprachmodell	System, das Wahrscheinlichkeiten für Wortfolgen erfasst, um Texte zu analysieren oder zu erzeugen
regelbasiertes System	arbeitet mit festen, von Menschen geschriebenen Regeln (z. B. ELIZA)
n-Gramm-Modell	statistisches Sprachmodell, das das nächste Wort aus den $n-1$ vorherigen Wörtern vorhersagt
Textkörperanalyse	Auszählen, wie oft Wortfolgen (n-Gramme) in einem Textkörper (Korpus) vorkommen
Textgenerierung	Erzeugen von Text, Wort für Wort (next word prediction)
Sampling	Auswahl des nächsten Wortes aus den möglichen Folgewörtern; die Sampling-Strategie (greedy, random, top-k) legt fest, wie ausgewählt wird
Halluzination	flüssig klingender, aber inhaltlich falscher oder erfundener Text

4.7.2 Kompetenz-Check

Kreuze ehrlich an, wie sicher du dich fühlst. Wo du „noch nicht“ ankreuzt, lies den passenden Abschnitt noch einmal und bearbeite seine Arbeitsaufträge.

Gymnasium Ottweiler

Ich kann …	sicher	teilweise	noch nicht
den Begriff Sprachmodell erläutern.			
regelbasierte Systeme (z. B. ELIZA) und n-Gramm-Modelle unterscheiden.			
in einem Text n-Gramm-Häufigkeiten auszählen und Wahrscheinlichkeiten berechnen.			
Sampling-Strategien (greedy, random, top-k) benennen und anwenden.			
erläutern, wie der erzeugte Text von den Trainingsdaten abhängt.			
beschreiben, wie sich die n-Gramm-Länge auf den erzeugten Text auswirkt.			
Halluzinationen anhand von n-Gramm-Modellen erklären.			
Unterschiede zwischen n-Gramm-Modellen und modernen LLMs beschreiben.			
Auswirkungen generativer KI auf Berufe recherchieren.			

Gymnasium Ottweiler