Ollama verstehen

Wenn man Ollama so konfigurieren will, dass ein Modell **viel Kontext verarbeiten** und **lange Antworten erzeugen** kann, stößt man sehr schnell auf Begriffe wie `num_ctx` und `num_predict`. Diese beiden Werte sind tatsächlich zentral — aber sie sind **nicht die ganze Geschichte**.

Denn in der Praxis hängt die Qualität und Länge von Antworten nicht nur davon ab, **was man einstellt**, sondern auch davon:

- **welches Modell** man verwendet,
- **welche Kontextlänge das Modell nativ unterstützt**,
- wie viel **RAM/VRAM** vorhanden ist,
- welche **Quantisierung** genutzt wird,
- und ob die Antworten trotz großer Kontexte **noch sinnvoll und stabil** bleiben.

Dieser Artikel erklärt die wichtigsten Ollama-Werte rund um Kontext und Antwortlänge, was sie technisch bedeuten, wie sie sich auf das Verhalten des Modells auswirken und welche sinnvollen Konfigurationen es für verschiedene Anwendungsfälle gibt.

---

1. Das Grundprinzip: Eingabetokens und Ausgabetokens

Bevor wir über Parameter sprechen, ist ein Konzept entscheidend:

Ein Sprachmodell arbeitet mit einem **Kontextfenster**. In diesem Fenster befinden sich alle Tokens, die das Modell aktuell „sehen“ kann. Dazu gehören z. B.:

- die Systemanweisung,
- die bisherigen Chat-Nachrichten,
- eingefügte Dokumente,
- dein aktueller Prompt,
- und je nach Ablauf auch Teile der bereits erzeugten Antwort.

Ein **Token** ist kein Wort, sondern ein kleines Textstück. Als grobe Faustregel gilt:

- 1 Token ≈ 0,75 Wörter auf Englisch,
- auf Deutsch oft ähnlich, manchmal etwas mehr,
- 1000 Tokens entsprechen grob einigen hundert bis etwa 750 Wörtern.

Wichtig ist: Das Modell hat **kein unbegrenztes Gedächtnis**. Es kann nur die Tokens berücksichtigen, die in sein Kontextfenster passen.

---

2. `num_ctx`: Wie viel Kontext das Modell sehen kann

Was bedeutet `num_ctx`?

`num_ctx` bestimmt die **maximale Kontextlänge**, also wie viele Tokens insgesamt dem Modell zur Verfügung stehen.

Das betrifft insbesondere:

- den aktuellen Prompt,
- Chat-Verlauf,
- eingefügte Texte oder Dokumente,
- und den restlichen Dialogkontext.

Kurz gesagt:

- **großes** `num_ctx` = mehr Text auf einmal verarbeitbar
- **kleines** `num_ctx` = weniger Gedächtnis, schnellere Begrenzung

Warum ist `num_ctx` so wichtig?

Wenn du z. B. möchtest, dass ein Modell:

- lange Dokumente analysiert,
- Code über viele Dateien hinweg berücksichtigt,
- in langen Chats frühere Aussagen erinnert,

- große Wissensblöcke im Prompt verarbeitet,

dann brauchst du ein ausreichend großes `num_ctx`.

Beispiel

Du gibst dem Modell:

- 20.000 Tokens Dokumentinhalt,
- 2.000 Tokens Instruktionen,
- 3.000 Tokens Chatverlauf.

Dann bist du schon bei **25.000 Tokens Eingabekontext**. Mit `num_ctx=8192` würde das nicht vollständig hineinpassen. Ein Teil müsste abgeschnitten werden oder der Prompt würde fehlschlagen bzw. intern gekürzt.

Was passiert bei zu kleinem `num_ctx`?

Ein zu kleines Kontextfenster führt oft zu:

- abgeschnittenem Chat-Verlauf,
- verlorenen Details aus früheren Nachrichten,
- schlechterer Kohärenz in langen Unterhaltungen,
- unvollständiger Dokumentenanalyse,
- „Vergessen“ von Vorgaben.

Das zeigt sich dann in Antworten wie:

- das Modell widerspricht sich später,
- es ignoriert frühe Anweisungen,
- es beantwortet nur den letzten Teil eines langen Inputs,
- es halluziniert mehr, weil ihm Kontext fehlt.

Was passiert bei sehr großem `num_ctx`?

Ein großes Kontextfenster ist nützlich, hat aber Kosten:

- **mehr RAM/VRAM-Verbrauch**
- oft **langsamere Verarbeitung**
- bei manchen Modellen schlechtere Qualität weit hinten im Kontext
- manche Modelle unterstützen große Kontexte technisch nur begrenzt sinnvoll

Wichtig: Ein Modell kann zwar manchmal **mit großem Kontext gestartet** werden, aber das heißt nicht automatisch, dass es diesen Kontext **qualitativ gut nutzen** kann.

Es gibt einen Unterschied zwischen:

1. **technisch möglich**
2. **vom Modell nativ trainiert/unterstützt**
3. **inhaltlich noch zuverlässig**

Das ist extrem wichtig.

---

3. `num_predict`: Wie lang die Antwort werden darf

Was bedeutet `num_predict`?

`num_predict` legt fest, wie viele **neue Tokens das Modell maximal erzeugen darf**.

Also:

- `num_ctx` = wie viel das Modell lesen kann
- `num_predict` = wie viel das Modell schreiben darf

Wenn du lange Antworten möchtest, ist `num_predict` der wichtigste direkte Parameter.

Beispiel

Wenn du `num_predict=256` setzt, dann kann die Antwort maximal etwa 256 Tokens lang sein. Das ist oft nur eine kurze bis mittlere Antwort.

Wenn du `num_predict=2048` setzt, dann kann die Antwort deutlich ausführlicher werden.

Wenn du sehr lange Artikel, Analysen oder Zusammenfassungen willst, sind Werte wie:

- 1024
- 2048
- 4096

oft realistischer.

Was passiert bei zu kleinem `num_predict`?

Dann wirkt das Modell oft so, als würde es:

- mitten im Satz abbrechen,
- zu knapp antworten,
- Aufzählungen nicht zu Ende führen,
- bei „schreibe einen ausführlichen Artikel“ trotzdem nur einen kurzen Text liefern.

In vielen Fällen ist das kein „Unwille“ des Modells, sondern schlicht eine harte Tokenbegrenzung.

Was passiert bei sehr großem `num_predict`?

Große Werte erlauben lange Antworten, aber auch hier gibt es Nebenwirkungen:

- längere Laufzeit,
- mehr Rechenaufwand,
- höheres Risiko von Wiederholungen,
- bei manchen Modellen größere Tendenz zum Abschweifen,
- mehr Kosten bei API-/Ressourcennutzung.

Außerdem heißt ein hoher Wert nicht, dass das Modell **immer** so lang antwortet. Er ist nur das **Maximum**.

Das Modell kann trotzdem kurz antworten, wenn:

- der Prompt nach Kürze klingt,

- die Antwort eigentlich kurz ist,
- ein Stop-Kriterium erreicht wird,
- die Sampling-Parameter eher kurze oder prägnante Antworten begünstigen.

---

4. Zusammenspiel von `num_ctx` und `num_predict`

Viele verstehen diese beiden Werte anfangs isoliert. In Wirklichkeit wirken sie zusammen.

Denkmodell

- `num_ctx` = wie viel „Gedächtnisplatz“ vorhanden ist
- `num_predict` = wie viel „Schreibplatz“ für die Antwort erlaubt wird

Typischer Fall 1: Großer Input, kurze Antwort

Du willst ein langes Dokument analysieren und nur eine knappe Zusammenfassung.

Dann brauchst du:

- **hohes** `num_ctx`
- aber nur **moderates** `num_predict`

Beispiel:

- `num_ctx = 32768`
- `num_predict = 512`

Typischer Fall 2: Mittlerer Input, lange Antwort

Du gibst eine überschaubare Aufgabe, willst aber einen langen Aufsatz.

Dann brauchst du:

- **mittleres** `num_ctx`
- **hohes** `num_predict`

Beispiel:

- `num_ctx = 8192`
- `num_predict = 2048` oder `4096`

Typischer Fall 3: Großer Input und lange Antwort

Das ist der anspruchsvollste Fall, z. B.:

- viele Dokumente einlesen,
- dann einen langen Bericht schreiben,
- mit Zitaten, Struktur und Begründungen.

Dann brauchst du beides hoch:

- **großes** `num_ctx`
- **großes** `num_predict`

Beispiel:

- `num_ctx = 32768` oder `65536`
- `num_predict = 2048` bis `4096+`

Aber genau hier steigen Speicherbedarf und Laufzeit stark.

---

5. Warum „mehr Kontext“ nicht immer automatisch

„besser“ ist

Das ist einer der wichtigsten Punkte.

Viele denken:
Je größer `num_ctx`, desto besser die Antwort.

Das stimmt nur teilweise.

Problem 1: Das Modell wurde vielleicht nicht für so viel Kontext trainiert

Ein Modell hat häufig eine **native oder empfohlene Kontextlänge**. Wenn man diese stark überschreitet, kann Folgendes passieren:

- das Modell beachtet nur frühe oder späte Teile zuverlässig,
- es „verwischt“ Informationen,
- die Relevanzbewertung wird schlechter,
- es reagiert inkonsistent.

Ein Modell kann technisch vielleicht mit 32k oder 64k gestartet werden, obwohl es bei 8k oder 16k am besten arbeitet.

Problem 2: Relevante Informationen gehen in der Masse unter

Auch wenn formal alles in den Kontext passt, heißt das nicht, dass das Modell alles gleich gut nutzt.

Wenn du 50 Seiten Kontext gibst, aber nur drei Sätze relevant sind, kann die Antwort schlechter werden als mit einem gut kuratierten, kompakten Prompt.

Problem 3: Performance sinkt

Mehr Kontext bedeutet meist:

- langsameres Prompt-Processing,
- oft mehr Speicherverbrauch,
- trägere Reaktionszeit.

Gerade lokal mit Ollama ist das sehr spürbar.

---

6. Warum „mehr `num_predict` “ nicht automatisch „bessere lange Antworten“ bedeutet

Ein hoher Wert ist nötig, aber nicht hinreichend.

Mögliche Probleme bei sehr langen Generationen

- Wiederholungen
- inhaltliches Kreisen
- Abschweifen
- sinkende Strukturqualität
- mehr Halluzinationen im späteren Verlauf

Je länger eine Antwort wird, desto wichtiger wird:

- ein guter Prompt,

- klare Gliederungsvorgaben,
- ggf. niedrigere Temperatur,
- explizite Anforderungen wie „in 8 Abschnitten“, „mit Zwischenüberschriften“, „ohne Wiederholungen“.

Praktischer Tipp

Wenn du sehr lange und gute Antworten willst, hilft oft mehr als bloß `num_predict` zu erhöhen:

Statt nur zu sagen:

> Schreib mir einen langen Text

besser:

> Schreibe einen strukturierten Artikel mit Einleitung, 6 Hauptabschnitten, Praxisbeispielen, Grenzen, Fazit. Vermeide Wiederholungen und führe jeden Punkt konkret aus.

So nutzt du das erlaubte Ausgabebudget sinnvoller.

---

7. Weitere wichtige Parameter, die Antworten beeinflussen

Auch wenn `num_ctx` und `num_predict` die Hauptrollen spielen, gibt es weitere Werte, die das Antwortverhalten stark beeinflussen.

`temperature`

Bestimmt, wie kreativ oder zufällig das Modell antwortet.

- **niedrig** (z. B. 0.1–0.3): präziser, nüchterner, stabiler
- **mittel** (z. B. 0.5–0.8): ausgewogen
- **hoch** (z. B. 0.9+): kreativer, aber riskanter und unsteter

Einfluss auf lange Antworten

Für lange fachliche Texte ist eine **zu hohe Temperatur** oft problematisch:

- mehr Abschweifungen,
- mehr Halluzinationen,
- mehr stilistische Unruhe.

Für sachliche, längere Ausgaben sind oft Werte um **0.2 bis 0.7** sinnvoll.

`top_k`

Begrenzt die Auswahl auf die wahrscheinlichsten nächsten Tokens.

- kleinerer Wert: konservativer
- größerer Wert: offener

Wirkung

Kann helfen, Antworten kontrollierter oder kreativer zu machen. Für nüchterne und längere Analysen eher moderat halten.

`top_p`

Alternative bzw. Ergänzung zu `top_k`.

Das Modell wählt aus den wahrscheinlichsten Tokens, bis eine kumulierte Wahrscheinlichkeit erreicht ist.

- kleinerer Wert: enger, konservativer
- größerer Wert: freier

Auch das beeinflusst, ob lange Antworten eher stabil oder eher driftend werden.

`repeat_penalty`

Reduziert Wiederholungen.

Gerade bei langen Antworten kann das wichtig sein, weil Modelle sonst dazu neigen, Formulierungen, Satzmuster oder Inhalte zu wiederholen.

Ein etwas erhöhter Wert kann helfen, aber zu viel kann auch unnatürliche Sprache erzeugen.

Stop-Sequenzen / `stop`

Mit Stop-Sequenzen kann man festlegen, wann die Ausgabe enden soll.

Das ist nützlich, wenn ein Modell sonst weiterredet, in neue Rollen springt oder Formatgrenzen ignoriert.

Aber Vorsicht:
Eine unpassende Stop-Sequenz kann lange Antworten **frühzeitig abschneiden**.

---

8. Speicher, Geschwindigkeit und Hardware

Großer Kontext ist vor allem ein **Hardware-Thema**.

Mehr Kontext kostet Speicher

Je größer `num_ctx`, desto mehr Speicher wird für den KV-Cache und die Verarbeitung benötigt. Das heißt:

- größere `num_ctx`-Werte brauchen deutlich mehr RAM/VRAM,
- größere Modelle verschärfen das Problem,
- lange Kontexte und lange Antworten zusammen sind besonders anspruchsvoll.

Auswirkungen in der Praxis

Wenn du lokal arbeitest, merkst du das oft so:

- das Modell startet langsamer,
- die erste Token-Ausgabe dauert länger,
- Prompt-Verarbeitung wird zäh,
- bei zu hohen Werten wird geswappt oder es kommt zu Fehlern,
- die Antwortgeschwindigkeit sinkt massiv.

Wichtiger praktischer Zusammenhang

Ein kleines Modell mit großem Kontext kann manchmal praktikabler sein als ein großes Modell mit riesigem Kontext.

Beispielhaft:

- Ein 7B- oder 8B-Modell mit 32k Kontext kann auf einem Desktop eher realistisch sein.
- Ein deutlich größeres Modell mit derselben Kontextlänge kann lokal schnell unpraktisch werden.

---

9. Modellgrenzen: Nicht jedes Modell ist für riesige

Kontexte geeignet

Das ist vielleicht der wichtigste praktische Rat überhaupt:

Ollama kann nur das freischalten, was Modell und Hardware sinnvoll hergeben.

Wenn ein Modell von Haus aus eher auf 4k, 8k oder 16k ausgelegt ist, bringt es oft wenig, einfach `num_ctx` extrem hochzusetzen.

Worauf du achten solltest

Beim gewählten Modell prüfen:

- empfohlene Kontextlänge,
- bekannte Langkontext-Fähigkeit,
- Community-Erfahrungen,
- Stabilität bei 32k/64k/128k,
- Speicherbedarf bei deiner Quantisierung.

Ein Modell mit offiziell guter Long-Context-Unterstützung ist für Dokumentanalyse oft viel wertvoller als ein Modell, das nur theoretisch groß konfigurierbar ist.

---

10. Typische Anwendungsfälle und sinnvolle Einstellungen

A. Langer Chat mit viel Gesprächsverlauf

Ziel:

- frühere Nachrichten erinnern,
- konsistent bleiben,
- über viele Runden hinweg zusammenhängend antworten.

Empfehlung:

- `num_ctx`: eher hoch, z. B. 16384 bis 32768
- `num_predict`: moderat, z. B. 512 bis 1024

Warum?
Der Chat-Verlauf ist wichtiger als riesige Einzelausgaben.

B. Dokumentanalyse großer Texte

Ziel:

- lange Texte lesen,
- gezielt zusammenfassen oder Fragen beantworten.

Empfehlung:

- `num_ctx`: hoch, z. B. 32768 oder mehr, falls Modell geeignet
- `num_predict`: 256 bis 1024, je nach gewünschter Antwortlänge

Warum?
Das Modell muss viel lesen, aber oft nicht extrem viel schreiben.

C. Lange Essays, Berichte, Blogartikel

Ziel:

- ausführliche, strukturierte Ausgaben erzeugen.

Empfehlung:

- `num_ctx`: mittel bis hoch, z. B. 8192 bis 16384

- `num_predict`: hoch, z. B. 2048 bis 4096

Warum?
Der eigentliche Schreibraum ist hier besonders wichtig.

D. Code-Assistenz mit größeren Projekten

Ziel:

- mehrere Dateien, APIs, Fehlerlogs und Anforderungen gleichzeitig berücksichtigen.

Empfehlung:

- `num_ctx`: hoch
- `num_predict`: mittel bis hoch

Warum?
Sowohl Kontext als auch Antwortlänge sind relevant. Besonders bei Refactoring, Architekturvorschlägen oder Datei-übergreifender Analyse.

---

11. Wie man Konfigurationen sinnvoll testet

Die beste Einstellung findet man selten nur theoretisch. Sinnvoll ist ein systematisches Vorgehen.

Schritt 1: Das Ziel definieren

Willst du vor allem:

- längere Antworten?
- längeren Chat-Verlauf?
- große Dokumente im Prompt?

- stabile Fachtexte?
- schnelle Reaktion?

Je nach Ziel verschiebt sich die optimale Konfiguration.

Schritt 2: Erst konservativ starten

Zum Beispiel:

- `num_ctx = 8192`
- `num_predict = 1024`

Dann prüfen:

- reicht der Kontext?
- wird die Antwort abgeschnitten?
- ist die Qualität stabil?
- wie schnell läuft es?

Schritt 3: Nur einen Wert auf einmal erhöhen

Erhöhe entweder:

- erst `num_ctx`, wenn Kontext fehlt,
- oder erst `num_predict`, wenn Antworten zu kurz sind.

Nicht alles gleichzeitig hochdrehen, sonst ist schwer erkennbar, was geholfen oder geschadet hat.

Schritt 4: Mit echten Aufgaben testen

Nicht nur mit Mini-Prompts. Nutze:

- dein echtes Dokument,
- deinen realen Chat-Verlauf,

- deine typischen Schreibaufgaben.

Nur dann siehst du, ob die Konfiguration wirklich passt.

---

12. Praktische Faustregeln

Wenn du große Dokumente verarbeiten willst:

Erhöhe zuerst **`num_ctx`**.

Wenn du längere Antworten willst:

Erhöhe zuerst **`num_predict`**.

Wenn Antworten trotz hohem `num_predict` kurz bleiben:

Dann liegt es oft am Prompt oder an Stop-Kriterien, nicht nur an der Token-Grenze.

Wenn das Modell frühere Infos vergisst:

Dann ist oft `num_ctx` zu klein oder der relevante Kontext wird im Chat verdrängt.

Wenn das Modell langsam wird oder abstürzt:

Dann ist `num_ctx` möglicherweise zu hoch für dein Modell oder deine Hardware.

Wenn lange Antworten schlechter werden:

Dann nicht nur `num_predict` erhöhen, sondern:

- Prompt besser strukturieren,
- Temperatur etwas senken,
- ggf. Ausgabe in Abschnitte aufteilen.

---

13. Beispielhafte Denkmuster statt „ein bester Wert“

Es gibt **keinen universell besten Wert** für `num_ctx` oder `num_predict`.

Stattdessen sind diese Denkmuster nützlich:

„Ich will, dass das Modell mehr erinnert“

→ `num_ctx` erhöhen

„Ich will, dass es nicht mitten im Artikel aufhört“

→ `num_predict` erhöhen

„Ich will lange Dokumente und dann ausführliche Ausgaben“

→ beide erhöhen, Hardware und Modellgrenzen beachten

„Ich will bessere Qualität bei langen Antworten“

→ nicht nur Tokenlimits anheben, sondern Sampling und Prompt verbessern

---

14. Typische Missverständnisse

Missverständnis 1:

„Großer Kontext heißt automatisch gute Langzeit-Erinnerung.“

Nicht unbedingt. Das Modell kann Informationen formal im Fenster haben, sie aber nicht optimal gewichten.

Missverständnis 2:

„Wenn ich `num_predict` riesig setze, bekomme ich automatisch lange Antworten.“

Nein. Es ist nur das Maximum. Der Prompt und das Modellverhalten entscheiden mit.

Missverständnis 3:

„Ich kann jedes Modell einfach auf 64k oder 128k Kontext stellen.“

Technisch vielleicht teilweise, aber sinnvoll und qualitativ stabil ist das nicht bei jedem Modell.

Missverständnis 4:

„Wenn Antworten abgeschnitten werden, brauche ich nur mehr `num_ctx`.“

Nicht unbedingt. Oft ist dann eher `num_predict` zu klein.

---

15. Konkrete Kurzempfehlungen

Wenn du mit Ollama arbeitest und einfach praktikabel starten willst:

Für allgemeine Nutzung

- `num_ctx`: 8192
- `num_predict`: 512 bis 1024

Für große Dokumente

- `num_ctx`: 16384 bis 32768
- `num_predict`: 512 bis 1024

Für lange Artikel/Erklärungen

- `num_ctx`: 8192 bis 16384
- `num_predict`: 2048 bis 4096

Für sehr anspruchsvolle Long-Context-Aufgaben

- `num_ctx`: so hoch wie Modell + Hardware sinnvoll erlauben
- `num_predict`: je nach gewünschter Ausgabelänge
- zusätzlich auf Modellwahl und Promptstruktur achten

---

16. Fazit

Die beiden wichtigsten Ollama-Werte für dein Anliegen sind:

- `num_ctx` für die **Menge an verarbeitbarem Kontext**
- `num_predict` für die **maximale Länge der Antwort**

Man kann sich das so merken:

- **`num_ctx` = lesen**
- **`num_predict` = schreiben**

Wenn du große Kontexte verarbeiten willst, musst du `num_ctx` erhöhen.
Wenn du lange Antworten möchtest, musst du `num_predict` erhöhen.
Wenn du beides willst, musst du beides erhöhen — aber dann steigen auch die Anforderungen an Modell, Speicher und Laufzeit.

Der wichtigste praktische Punkt ist dabei:
Nicht nur größer einstellen, sondern passend einstellen.

Denn ein riesiger Kontext bringt wenig, wenn:

- das Modell dafür nicht gut geeignet ist,
- die Hardware überlastet wird,
- der Prompt schlecht strukturiert ist,

- oder die Antwortqualität bei langen Generationen sinkt.

Die beste Konfiguration ist deshalb immer ein Kompromiss aus:

- Kontextbedarf,
- gewünschter Antwortlänge,
- Modellfähigkeit,
- Stabilität,
- und verfügbarer Hardware.

Revision #1
Created 2026-06-08 17:34:11 UTC by art10m
Updated 2026-06-08 17:34:34 UTC by art10m