

WHITEPAPER

Tokenverbrauch im Griff

Strategien zur Kostenkontrolle beim Einsatz von KI-Assistenten in Unternehmen

Vom Pauschaltarif zur verbrauchsbasierten Abrechnung und was das für Budgets, Workflows und Tool-Auswahl bedeutet.

Herausgeber: twenty5ai – Sebastian Oczachowski
Version 1.0 · Juni 2026

Vertraulichkeit: öffentlich

Management Summary

Mit der Umstellung von GitHub Copilot auf eine **token-basierte Abrechnung** zum 1. Juni 2026 ist ein Thema in der Breite angekommen, das bislang vor allem Plattform-Teams beschäftigte: Die Kosten eines KI-Assistenten hängen nicht mehr an einer planbaren Pauschale, sondern am tatsächlichen Tokenverbrauch.

Für Unternehmen bedeutet das einen Wechsel der Kalkulationslogik. Budgets, die zuvor pro Nutzer und Monat fix waren, werden variabel und workload-abhängig. Gemeldet werden für intensive, agentische Anwendungsfälle Kostensteigerungen im Bereich **des Zehn- bis Fünfzigfachen**. Wer KI-Assistenten produktiv einsetzt, braucht daher eine bewusste Strategie zur Steuerung des Tokenverbrauchs.

Dieses Whitepaper ordnet das Problem ein, erklärt den zugrunde liegenden Kostenmechanismus und bewertet die gängigen Spar-Ansätze von Kontextstrategien (langer Chat vs. periodischer Reset vs. Verdichtung) über Memory-Funktionen und Prompt-Caching bis zu Modellwahl und Retrieval. Eine Modellrechnung zeigt: Der mit Abstand teuerste Reflex ist es, ein einzelnes Kontextfenster immer weiter zu füllen.

Die drei wichtigsten Erkenntnisse auf einen Blick:

- Der Mechanismus ist kumulativ. In einem fortlaufenden Chat wird die gesamte bisherige Historie in jeder Runde erneut als Input verarbeitet. Die Kosten wachsen überproportional (quadratisch), nicht linear.
- Kontext ist das teuerste Gut. Verdichtung (Summarization, Memory, Kontext-Deckelung) schlägt das ungebremste Füllen des Kontextfensters in der Modellrechnung um rund 70 %.
- Die größten Hebel sind günstig. Modellwahl, Prompt-Caching und Output-Disziplin liefern hohe Einsparungen bei geringem Umsetzungsaufwand und daher gehören sie zuerst angegangen.

Inhalt

Management Summary.....	2
Inhalt	3
1 Ausgangslage: Vom Pauschaltarif zur verbrauchs-basierten Abrechnung	4
1.1 Der Auslöser: GitHub Copilot wechselt das Abrechnungsmodell.....	4
1.2 Warum das ein strukturelles Problem ist – nicht nur ein Copilot-Thema.....	5
1.3 Leitfrage, These und Aufbau	6
2 Was ist ein Token? Vom Wort zum abgerechneten Baustein.....	7
2.1 Vom Wort zum Token: die Faustregeln.....	7
2.2 Warum die Token-Zahl schwankt: Sprache, Code und Sonderzeichen.....	8
2.3 Was genau gezählt wird – und der Unterschied zur Request-Abrechnung	9
3 Grundlagen: Wie aus Tokens laufende Kosten werden.....	10
3.1 Der kumulative Effekt: Jede Runde sendet den ganzen Verlauf erneut	10
4 Ansätze zur Tokenersparnis	11
4.1 Kontextstrategien: langer Chat, neuer Chat oder Verdichtung.....	11
4.2 Memory-Funktionen	12
4.3 Prompt-Caching	12
4.4 Modellwahl und Routing	13
4.5 Retrieval statt Context-Stuffing (RAG)	13
4.6 Output-Disziplin und Reasoning-Effort	13
5 Bewertung: Hebelwirkung und Umsetzungsaufwand	14
6 Handlungsempfehlungen für Unternehmen	15
7 Fazit	16
Anhang A Sofort einsetzbare Vorlage: Token-Disziplin per Instruktionsdatei.....	17
A.1 So setzen Sie die Vorlage ein.....	17
A.2 Die Vorlage	17
A.3 Welcher Schritt welchen Mehrwert bringt	18
Methodik, Grenzen und Quellen.....	19

1 Ausgangslage: Vom Pauschaltarif zur verbrauchsasierten Abrechnung

1.1 Der Auslöser: GitHub Copilot wechselt das Abrechnungsmodell

Lange Zeit folgten KI-Assistenten in Unternehmen einer einfachen Preislogik: ein fester Betrag pro Nutzer und Monat, gegebenenfalls ergänzt um ein Kontingent an „Anfragen“. Eine Anfrage war eine Anfrage – unabhängig davon, ob im Hintergrund hundert oder hunderttausend Tokens verarbeitet wurden. Diese Abstraktion machte Budgets planbar und verbarg den eigentlichen Kostentreiber vor den Anwendenden.

Mit dem 1. Juni 2026 hat GitHub dieses Modell für Copilot abgelöst. An die Stelle der bisherigen „Premium Request Units“ trat eine verbrauchsasierte Abrechnung über „GitHub AI Credits“, die sich direkt am Tokenverbrauch orientiert – Input-, Output- und zwischengespeicherte (cached) Tokens, jeweils zu den veröffentlichten Preisen des gewählten Modells. Die Grundpreise der Pläne blieben unverändert, definieren nun aber ein monatliches Creditguthaben statt einer Obergrenze. Einfache Code-Vervollständigungen bleiben kostenfrei; Chat, agentische Sessions und Code-Reviews ziehen hingegen Credits.

Was sich konkret geändert hat

- Bezugsgröße: statt einer Anfrage zählt nun der verarbeitete Token – Input, Output und Cache.
- Wegfall des Sicherheitsnetzes: das frühere Ausweichen auf ein günstigeres Modell nach Verbrauch des Kontingents entfällt; sind die Credits aufgebraucht, stoppen die Premium-Funktionen.
- Modellpreis wird sichtbar: die Wahl des Modells wirkt sich nun unmittelbar auf die Kosten aus (siehe Abschnitt 4.4).

Auslaufende Jahrespläne und an die Nutzung gebundene Budget-Kontrollen verstärken den Effekt zusätzlich.

Die Reaktionen aus der Entwickler-Community fielen scharf aus. Für intensive, agentische Workflows – also Sitzungen, in denen der Assistent eigenständig viele Dateien liest, ändert und testet wurden Kostensteigerungen vom Zehn- bis Fünfzigfachen berichtet. Einzelne, öffentlich dokumentierte Praxisfälle reichen von einer Verteuerung von rund 29 auf 750 US-Dollar bis hin zu 50 auf 3.000 US-Dollar im Monat. Diese Werte sind keine Durchschnitte, sondern Spitzenlasten. Sie illustrieren jedoch die neue Spannweite, mit der Budgets nun umgehen müssen.

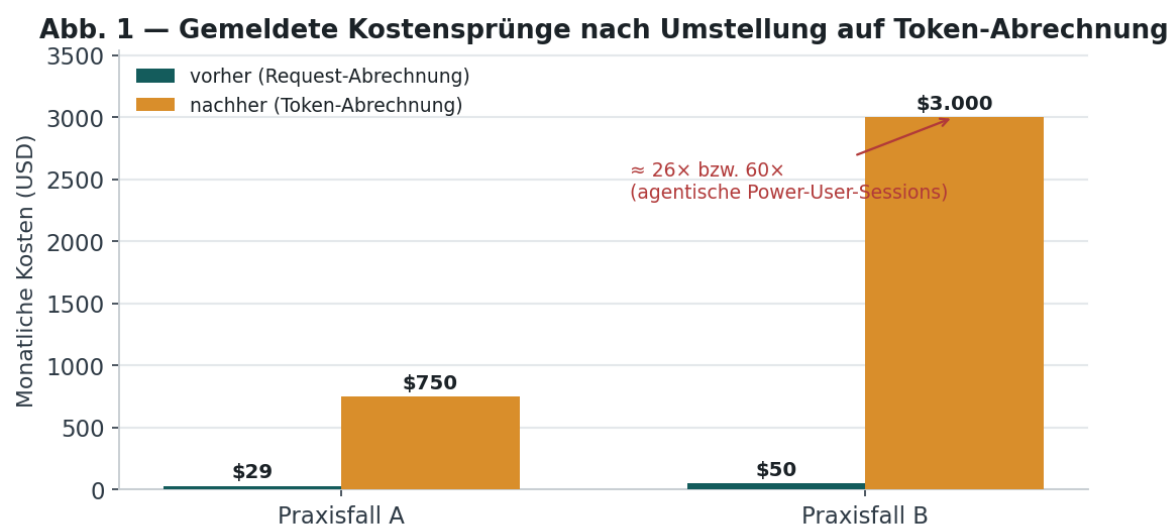


Abbildung 1: Gemeldete Kostensprünge einzelner Power-User nach der Umstellung auf token-basierte Abrechnung.
Quelle: Branchenberichte Mai/Juni 2026.

1.2 Warum das ein strukturelles Problem ist – nicht nur ein Copilot-Thema

Der Wechsel bei Copilot ist kein Einzelfall, sondern Symptom einer Marktbewegung. Alle großen Modellanbieter rechnen längst nach Tokens ab; die Pauschal- und Anfrage-Modelle der Endprodukte waren eine Abstraktionsschicht darüber, die angesichts steigender Infrastrukturkosten und zunehmend agentischer Nutzung wirtschaftlich unter Druck geriet. Es ist daher absehbar, dass weitere Anbieter ihre Abrechnung enger an den realen Verbrauch koppeln.

Für Unternehmen verschiebt sich damit die Kalkulationslogik grundlegend:

Dimension	Pauschal-/Anfrage-Modell	Token-/Verbrauchs-Modell
Kostenbasis	Fix pro Nutzer/Monat	Variabel nach Tokenverbrauch
Planbarkeit	Hoch, im Voraus bekannt	Workload-abhängig, schwankend
Kostentreiber	verborgen	Kontextlänge, Modell, Output
Steuerungshebel	Lizenzanzahl	Prompt-, Kontext- & Tooldesign
Verantwortung	Einkauf / IT	zusätzlich: Team & Anwender

Die entscheidende Konsequenz: Tokenkosten werden zu einer Größe, die nicht mehr allein im Einkauf, sondern im täglichen Arbeitsverhalten entschieden wird. Wer effizient promptet, Kontext bewusst verwaltet und das passende Modell wählt, zahlt einen Bruchteil dessen, was unbedachte Nutzung kostet. Genau hier setzen die folgenden Kapitel an.

1.3 Leitfrage, These und Aufbau

Dieses Whitepaper geht strukturiert einer zentralen Frage nach und beantwortet sie über die folgenden Kapitel hinweg:

Leitfrage

Welche Maßnahmen senken den Tokenverbrauch beim produktiven Einsatz von KI-Assistenten am wirksamsten und wie lassen sie sich ohne Einbußen bei der Ergebnisqualität in den Arbeitsalltag überführen?

These. Der größte Kostentreiber ist nicht die Wahl eines einzelnen Modells, sondern der unkontrolliert mitwachsende Kontext. Die wirksamsten Gegenmaßnahmen z.B. bewusste Modellwahl, Prompt-Caching sowie Output- und Kontextdisziplin sind zugleich diejenigen mit dem geringsten Umsetzungsaufwand. Diese These wird im Verlauf hergeleitet, mit einer Modellrechnung untermauert und in eine konkrete Vorlage überführt.

Aufbau:

Kapitel 2 klärt den Begriff des Tokens und die Umrechnung von Wörtern und Zeichen

Kapitel 3 erläutert den Kostenmechanismus, der dem Tokenverbrauch im Dialog zugrunde liegt

Kapitel 4 stellt die einzelnen Spar-Ansätze vor

Kapitel 5 bewertet sie nach Wirkung und Aufwand

Kapitel 6 leitet daraus ein gestuftes Vorgehen ab

Kapitel 7 fasst die Antwort auf die Leitfrage zusammen

Anhang A überführt die Erkenntnisse schließlich in eine sofort einsetzbare Instruktionsdatei

2 Was ist ein Token? Vom Wort zum abgerechneten Baustein

Bevor wir über das Sparen sprechen, lohnt der Blick auf die abgerechnete Einheit selbst. Denn genau hier liegt der Bruch zum alten Modell: Früher war die Bezugsgröße die Anfrage. Ob ein kurzer Satz oder ein 500-Zeilen-Code-Block übergeben wurde, spielte für die Abrechnung keine Rolle. Heute zählt der Token, und damit wird die Länge und Beschaffenheit jeder Ein- und Ausgabe unmittelbar kostenrelevant.

2.1 Vom Wort zum Token: die Faustregeln

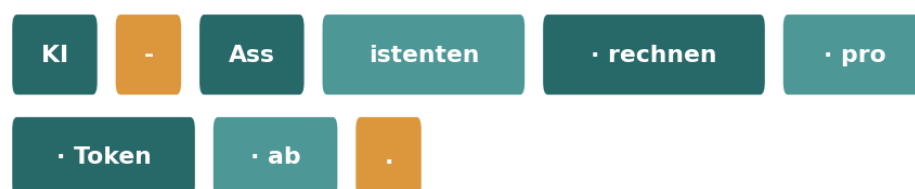
Ein Sprachmodell verarbeitet Text nicht als Wörter oder Buchstaben, sondern als „Tokens“ – Bausteine, die meist kleiner als ein Wort und größer als ein einzelner Buchstabe sind. Häufige Wörter werden zu einem einzigen Token; seltenere oder lange Wörter werden in mehrere Wortteile (Subwords) zerlegt. Dieses Prinzip der Subword-Zerlegung erlaubt es dem Modell, auch unbekannte Wörter, Eigennamen oder Tippfehler zu verarbeiten, ohne für jedes denkbare Wort einen eigenen Eintrag vorhalten zu müssen.

Für die grobe Abschätzung haben sich für englischsprachigen Text zwei Faustregeln etabliert:

- **≈ 4 Zeichen** entsprechen rund einem Token.
- **≈ 0,75 Wörter** entsprechen einem Token – anders gesagt: 100 Tokens sind etwa 75 Wörter.

Diese Werte sind Näherungen und gelten am ehesten für Englisch. Für andere Sprachen und insbesondere für Programmcode weichen sie deutlich ab (siehe Abschnitt 2.2). Auch Leerzeichen, Satzzeichen und Ziffern sind Tokens: Ein führendes Leerzeichen wird dabei häufig an das folgende Wort angehängt und zählt mit. Die folgende Abbildung zeigt schematisch, wie ein einfacher Satz in einzelne Tokens zerfällt.

Abb. 2 — Wie ein Satz in Tokens zerlegt wird (schematisch)



Satz: „KI-Assistenten rechnen pro Token ab.“

6 Wörter · 35 Zeichen · ≈ 9 Tokens („·“ markiert ein führendes Leerzeichen)

Abbildung 2: Schematische Zerlegung eines Satzes in Tokens. Häufige Wörter bleiben ganz, seltene werden in Wortteile zerlegt; Leerzeichen und Satzzeichen zählen mit. Die genaue Zerlegung hängt vom jeweiligen Tokenizer ab.

Das anschauliche Ergebnis: Aus sechs Wörtern und 35 Zeichen werden in diesem Beispiel rund neun Tokens. Wer den Tokenverbrauch grob einschätzen will, kann sich an der Zeichenzahl orientieren. Ein verlässliches Gefühl entsteht aber erst, wenn man die Einflussfaktoren kennt.

2.2 Warum die Token-Zahl schwankt: Sprache, Code und Sonderzeichen

Dieselbe Information kann sehr unterschiedlich viele Tokens kosten. Drei Faktoren sind besonders relevant:

- **Sprache:** Die gängigen Tokenizer sind stark auf Englisch optimiert. Deutsch benötigt im Schnitt spürbar mehr Tokens pro Wort. Untersuchungen zeigen Größenordnungen von rund 2,1 Tokens pro Wort für Deutsch gegenüber etwa 1,4 für Englisch. Ursache sind unter anderem die langen Komposita: Ein einzelnes deutsches Wort wie „Donaudampfschiffahrtsgesellschaft“ wird in viele Wortteile zerlegt. Auch Zahlwörter sind aufschlussreich – „neunzehnhundertvierundachtzig“ benötigt rund 11 Tokens, das englische „nineteen eighty four“ nur etwa 5.
- **Code:** Quelltext ist besonders token-dicht. Einrückungen, Klammern, Operatoren, Sonderzeichen und wiederkehrende Strukturen erzeugen viele Tokens pro sichtbarem Zeichen. Gerade hier wirkt sich die Umstellung aus: Ein langer Code-Block, der früher als eine Anfrage zählte, schlägt nun mit seiner vollen Tokenmenge zu Buche.
- **Tokenizer:** Es gibt nicht „den einen“ Tokenizer. Verschiedene Modelle nutzen verschiedene Tokenizer; derselbe Text kann je nach Modell unterschiedlich viele Tokens ergeben. Token-Schätzungen sind daher immer modellabhängig.

Für die Praxis bedeutet das: Deutschsprachige Inhalte und umfangreicher Code sind tendenziell teurer als ein vergleichbarer englischer Fließtext. Diese Faktoren sollten in die Kalkulation einfließen, etwa bei der Frage, ob ein großes Dokument vollständig oder nur in relevanten Auszügen übergeben wird (siehe Abschnitt 4.5).

Worked Example – dieselbe Aussage, sehr unterschiedliche Token-Zahl

Die Werte sind gerundete Richtwerte zur Veranschaulichung; die exakte Zahl hängt vom Tokenizer ab.

Beispiel	Wörter	Zeichen	≈ Tokens	Tokens/Wort
„Guten Morgen“ (Deutsch)	2	12	≈ 3	1,5
„Good morning“ (Englisch)	2	12	≈ 2	1,0
„Donaudampfschiffahrts-gesellschaft“	1	34	≈ 9	9,0
„neunzehnhundertvierund-achtzig“	1	29	≈ 11	11,0
Codezeile: <code>if (x > 0) { y = 1; }</code>	—	24	≈ 13	—

2.3 Was genau gezählt wird – und der Unterschied zur Request-Abrechnung

Abgerechnet werden in der Regel drei Arten von Tokens:

- Input-Tokens: alles, was an das Modell gesendet wird: Systemvorgaben, bisheriger Gesprächsverlauf, die aktuelle Frage und beigefügte Dokumente
- Output-Tokens: die Antwort des Modells. Sie sind pro Token meist deutlich teurer als Input-Tokens
- Cached-Tokens: wiederverwendete, zwischengespeicherte Eingabeteile, die stark vergünstigt verrechnet werden (siehe Abschnitt 4.3)

Der entscheidende Unterschied

Im alten, **request-basierten** Modell war die Bezugsgröße die Anfrage. Eine Frage war eine Anfrage, unabhängig davon, ob sie zehn oder zehntausend Tokens umfasste. Länge, Sprache und Code-Anteil einer Eingabe waren für die Abrechnung **irrelevant**.

Im **token-basierten** Modell ist genau das Gegenteil der Fall: Jeder zusätzliche Token, ob im Prompt, im mitgesendeten Kontext oder in der Antwort, schlägt sich direkt in den Kosten nieder. Damit wird das Verständnis dessen, was ein Token ist, zur Grundlage jeder Kostensteuerung.

3 Grundlagen: Wie aus Tokens laufende Kosten werden

Nachdem geklärt ist, was ein Token ist und was abgerechnet wird, betrachten wir den Mechanismus, der die laufenden Kosten im Dialog antreibt. Er ist die Grundlage für alle Spar-Ansätze des folgenden Kapitels.

3.1 Der kumulative Effekt: Jede Runde sendet den ganzen Verlauf erneut

Der wichtigste und am häufigsten übersehene Punkt: Ein Sprachmodell hat kein Gedächtnis zwischen einzelnen Aufrufen. Damit ein Chat „sich erinnert“, wird in jeder Runde der gesamte bisherige Gesprächsverlauf erneut als Input mitgeschickt. Die zehnte Frage in einem Chat trägt also nicht nur ihre eigenen Tokens, sondern bezahlt die erneute Verarbeitung der neun vorangegangenen Runden gleich mit.

Kernaussage

Die Kosten eines fortlaufenden Chats wachsen nicht linear, sondern **überproportional (näherungsweise quadratisch)** mit der Länge. Jeder zusätzliche Turn verteuert alle nachfolgenden Turns, weil das Kontextfenster mitwächst und vollständig neu verarbeitet wird.

Das Kontextfenster ist dabei die Obergrenze dessen, was ein Modell gleichzeitig verarbeiten kann. „Das Fenster voll auszunutzen“ ist technisch möglich, kostet aber genau deshalb am meisten, weil dieser volle Kontext bei jeder weiteren Interaktion erneut bezahlt wird. Aus diesem Mechanismus ergeben sich unmittelbar die im nächsten Kapitel diskutierten Strategien.

4 Ansätze zur Tokenersparnis

Die folgenden Ansätze adressieren unterschiedliche Stellhebel. Sie schließen sich nicht aus, sondern entfalten ihre Wirkung in Kombination. Wir behandeln sie von der Kontextverwaltung über das Caching bis zur Modell- und Output-Ebene.

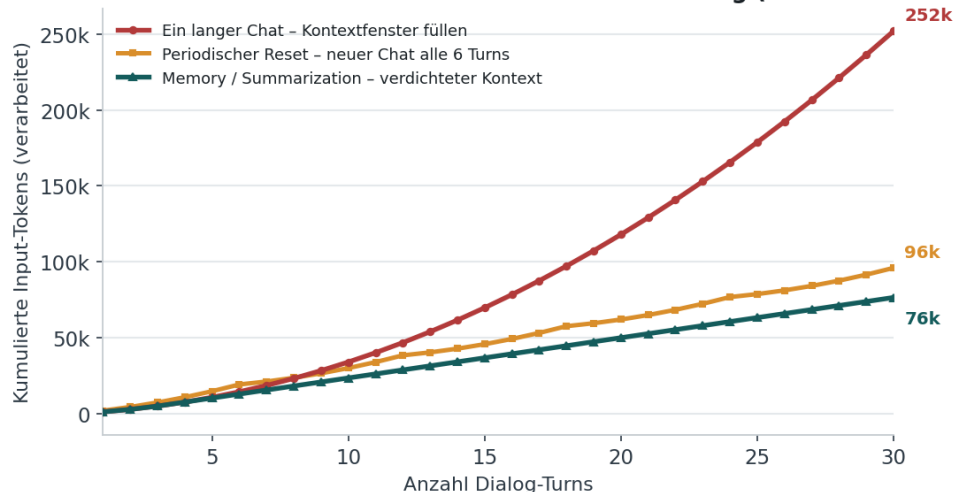
4.1 Kontextstrategien: langer Chat, neuer Chat oder Verdichtung

Aus dem kumulativen Effekt folgt die erste und folgenreichste Entscheidung: Wie wird der Gesprächskontext über die Zeit verwaltet? Drei Grundmuster lassen sich unterscheiden:

- **Ein langer Chat (Kontextfenster füllen):** alles bleibt in einer Sitzung. Maximaler Kontext, maximaler Komfort, aber überproportional steigende Kosten, weil der wachsende Verlauf in jeder Runde neu verarbeitet wird.
- **Periodischer Reset (neuer Chat):** regelmäßig einen frischen Chat starten. Setzt den Kontext auf nahezu null zurück und senkt die Kosten deutlich. Erkauft wird das mit dem Verlust an Kontext und dem Aufwand, relevante Informationen erneut einzubringen (Re-Primer).
- **Verdichtung (Memory / Summarization):** den Verlauf laufend zu einer kompakten Zusammenfassung eindampfen oder auf eine feste Obergrenze deckeln. Kombiniert niedrige Kosten mit erhaltenem Kontext. Daher der überzeugendste, aber technisch anspruchsvollste Weg.

Die folgende Modellrechnung stellt die drei Muster über 30 Dialog-Runden gegenüber. Sie ist illustrativ und beruht auf offengelegten Annahmen (siehe Bildunterschrift); die zugrunde liegende Token-Mathematik ist jedoch deterministisch und bildet das reale Verhalten qualitativ korrekt ab.

Abb. 3 — Kumulierter Token-Verbrauch über eine Sitzung (Modellrechnung)



Modellannahmen: System-Prefix 1.000, User 150, Assistant 350 Tokens/Turn; Re-Primer 800 Tokens; Summarization-Deckel 1.500 Tokens. Illustrativ, keine Messung.

Abbildung 3: Kumulierter Input-Tokenverbrauch über eine Sitzung nach Kontextstrategie (Modellrechnung). Der „lange Chat“ wächst überproportional; Reset und Verdichtung verlaufen annähernd linear.

Das Ergebnis ist eindeutig: Über 30 Runden verarbeitet der einzelne lange Chat in diesem Modell rund 252.000 Input-Tokens, der periodische Reset etwa 96.000 und die Verdichtungs-Strategie rund 76.500 – eine Reduktion von etwa 70 % gegenüber dem ungebremsen Kontextfenster. Wichtig für die Praxis:

- Der Reset spart viel, kostet aber Kontext. Er eignet sich, wenn Aufgaben klar abgegrenzt sind und wenig Vorwissen erneut benötigt wird.

- Verdichtung ist der Königsweg, erzeugt aber selbst Kosten: Das Zusammenfassen ist ein zusätzlicher Modellaufruf mit Output-Tokens. In langen Sitzungen amortisiert sich das schnell.

Praktische Faustregel: Lange, ausufernde Chats für umfangreiche Aufgaben vermeiden; stattdessen Aufgaben in abgegrenzte Sitzungen schneiden und, wo verfügbar, Verdichtungs- bzw. Memory-Mechanismen nutzen.

4.2 Memory-Funktionen

Memory-Funktionen heben relevante Informationen aus dem flüchtigen Chatverlauf in einen persistenten, sitzungsübergreifenden Speicher. Der entscheidende ökonomische Vorteil liegt nicht primär im Komfort, sondern in der Token-Bilanz: Statt in jeder neuen Sitzung den vollständigen Kontext erneut aufzubauen oder den alten Chat fortzuschleppen, wird nur ein kuratierter, verdichteter Auszug bereitgestellt.

Memory richtig einordnen

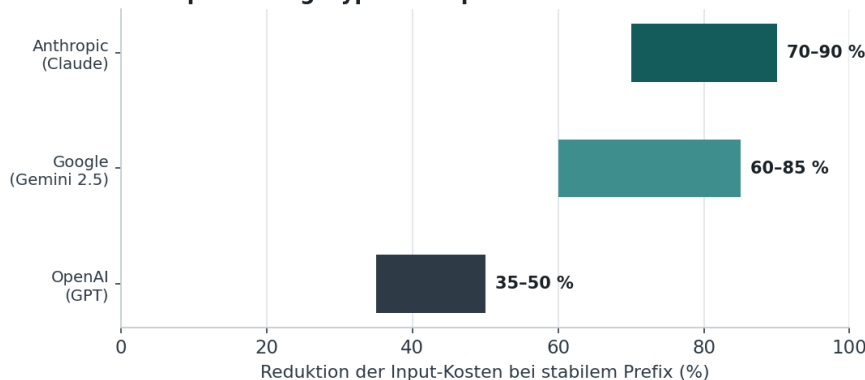
- Vorteil: persistenter Kontext ohne wiederholtes Re-Priming; vermeidet das teure Mitschleppen langer Verläufe.
- Vorsicht: auch Memory wird als Input mitgesendet und kostet Tokens. Unkontrolliert wachsender Speicher verkehrt den Spareffekt ins Gegenteil.

Empfehlung: Memory bewusst kuratieren. Wenige, hochrelevante Fakten statt vollständiger Protokolle.

4.3 Prompt-Caching

Wiederkehrende, identische Eingabeteile, etwa umfangreiche Systemanweisungen, Tool-Definitionen oder ein fester Dokumentenkontext, müssen nicht bei jeder Anfrage neu verarbeitet werden. Prompt-Caching speichert den internen Zustand für einen stabilen Prefix und liest ihn bei Folgeanfragen stark vergünstigt aus, ohne Qualitätsverlust. Voraussetzung ist, dass der Prefix Token für Token identisch bleibt: Eine einzige Änderung darin bricht den Cache ab dieser Stelle.

Abb. 4 — Prompt-Caching: typische Input-Kostenreduktion bei stabilen Prefixes



Quelle: Provider-Dokumentation & Branchenanalysen 2026. Gilt für wiederverwendete Input-Prefixes; Output-Tokens bleiben voll bepreist. Realwert hängt von der Cache-Trefferquote ab.

Abbildung 4: Typische Reduktion der Input-Kosten bei stabilen Prefixes je Anbieter. Gilt für wiederverwendete Eingabeteile; Output-Tokens bleiben voll bepreist.

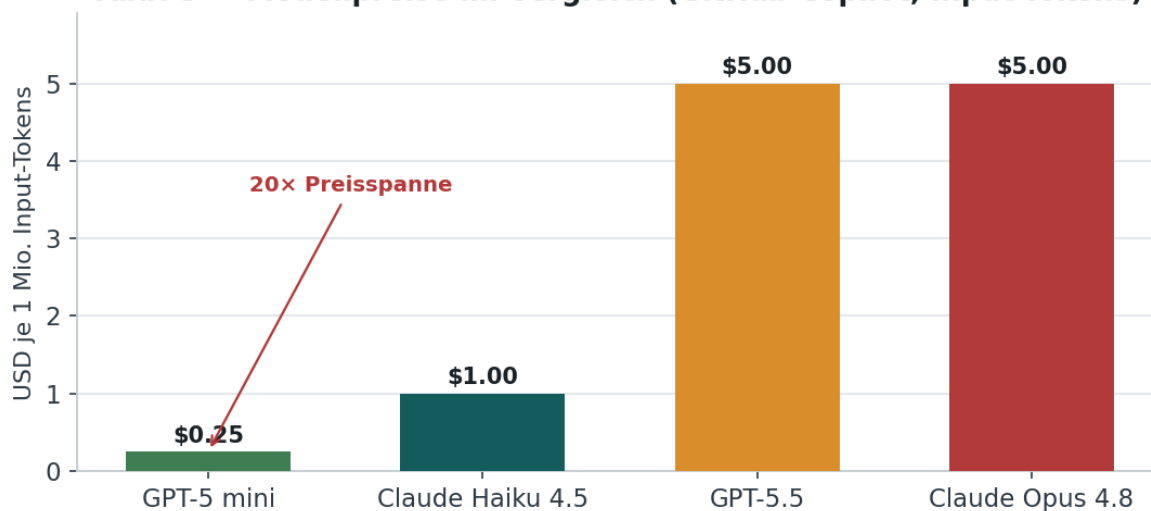
Die erzielbaren Einsparungen sind erheblich. Bei stabilen, umfangreichen Prefixes fallen die Input-Kosten je nach Anbieter um rund 35 bis 90 %. Der real erreichte Wert hängt von der Cache-Trefferquote ab: Nur tatsächlich wiederverwendete Prefixes sparen. Caching ist damit

einer der wirksamsten und zugleich am leichtesten zu aktivierenden Hebel und wird in der Praxis aber häufig ungenutzt gelassen. Wichtig: Es senkt ausschließlich Input-Kosten; teure Output-Tokens bleibt es schuldig.

4.4 Modellwahl und Routing

Seit der Modellpreis sichtbar geworden ist, ist das Greifen zum jeweils stärksten Modell nicht mehr kostenneutral. Die Preisspanne ist beträchtlich. Zwischen einem leichtgewichtigen und einem Hochleistungsmodell kann sie beim Input das Zwanzigfache betragen.

Abb. 5 — Modellpreise im Vergleich (GitHub Copilot, Input-Tokens)



Quelle: GitHub Copilot Modell-Preisliste 2026 (Auswahl). Output-Tokens werden gesondert und höher bepreist.

Abbildung 5: Input-Preise ausgewählter Modelle je 1 Mio. Tokens (GitHub Copilot). Output-Tokens werden gesondert und höher bepreist.

Die Konsequenz ist ein bewusstes Routing: einfache, gut strukturierte Aufgaben an günstige Modelle, komplexe mehrstufige Aufgaben an leistungsfähigere. Viele Plattformen bieten dafür einen Automatikmodus, der Anfragen je nach Anforderung an das passende Modell leitet. Für Teams gilt: Nicht das stärkste Modell als Standard setzen, sondern den Modelleinsatz an die Aufgabe knüpfen. Dies ist, gemeinsam mit Caching, der Hebel mit dem besten Verhältnis aus Wirkung und Aufwand.

4.5 Retrieval statt Context-Stuffing (RAG)

Ein verbreitetes Muster ist es, ganze Dokumente oder vollständige Repositories in den Kontext zu kippen, damit das Modell „alles weiß“. Genau das treibt die Input-Tokens in die Höhe. Retrieval-Augmented Generation (RAG) kehrt das Prinzip um: Statt alles mitzusenden, werden nur die für die konkrete Frage relevanten Ausschnitte gezielt herausgesucht und beigelegt.

Der Aufbau einer Retrieval-Pipeline ist aufwändiger als die übrigen Maßnahmen, lohnt sich aber bei großen, stabilen Wissensbeständen und ergänzt sich hervorragend mit Caching, da der Systemkontext stabil bleibt und die variablen, abgerufenen Ausschnitte klein gehalten werden.

4.6 Output-Disziplin und Reasoning-Effort

Zwei oft unterschätzte Hebel betreffen die Antwortseite. Erstens die Output-Disziplin: Da Output-Tokens pro Stück teurer sind als Input, lohnt es sich, das gewünschte Format und die Länge der Antwort vorzugeben. Knappe, strukturierte Antworten statt ausufernder Fließtexte. Zweitens der Reasoning-Effort: Viele aktuelle Modelle erlauben es, die Intensität des „Nachdenkens“ zu

steuern (etwa niedrig, mittel, hoch). Höhere Stufen erzeugen mehr interne Tokens und damit höhere Kosten; für einfache Aufgaben ist eine niedrigere Stufe oft ausreichend und deutlich günstiger.

5 Bewertung: Hebelwirkung und Umsetzungsaufwand

Nicht jede Maßnahme lohnt sich für jede Organisation gleichermaßen. Die folgende Einordnung positioniert die Ansätze nach ihrer Hebelwirkung auf die Tokenkosten und dem Aufwand der Umsetzung. Sie ist als Entscheidungshilfe gedacht; die konkrete Wirkung bleibt workload-abhängig.

Abb. 6 — Hebelwirkung vs. Umsetzungsaufwand der Maßnahmen (Einordnung)

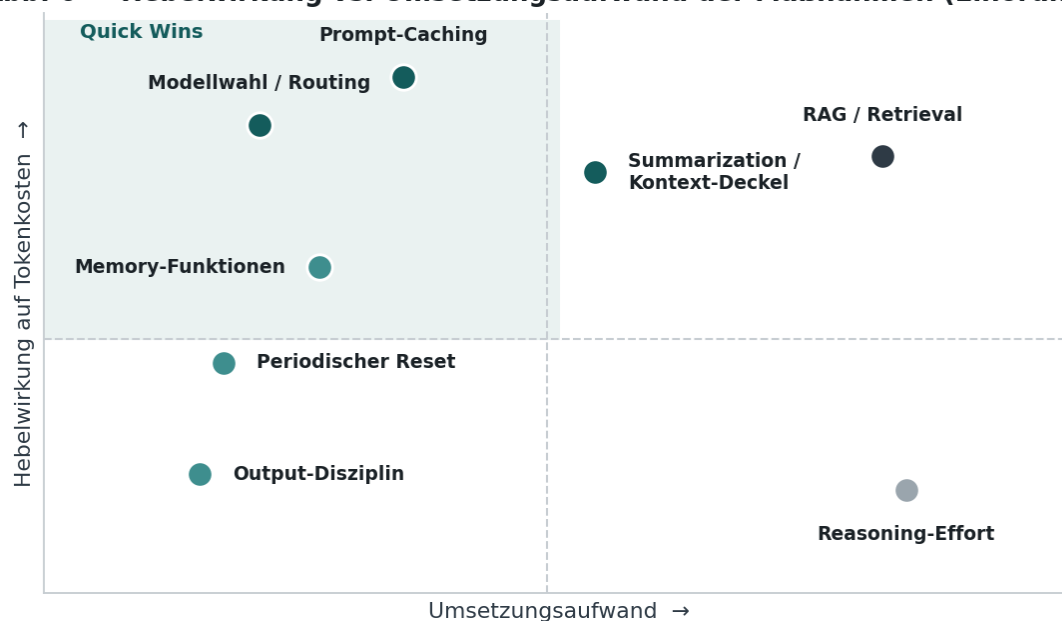


Abbildung 6: Einordnung der Maßnahmen nach Hebelwirkung und Umsetzungsaufwand. „Quick Wins“ verbinden hohe Wirkung mit geringem Aufwand.

Die Lesart ist einfach: Maßnahmen oben links („Quick Wins“) verbinden hohe Wirkung mit geringem Aufwand und gehören zuerst angegangen. Allen voran bewusste Modellwahl, Prompt-Caching und Output-Disziplin. Aufwändigere Bausteine wie RAG entfalten ihre volle Wirkung erst bei entsprechender Datenlage und sind als zweite Stufe sinnvoll.

Maßnahme	Hebel	Aufwand	Wann besonders sinnvoll
Modellwahl / Routing	hoch	gering	durchgängig, jede Organisation
Prompt-Caching	hoch	gering	stabile, große Prefixes / hohe Frequenz
Summarization / Deckel	hoch	mittel	lange, fortlaufende Sitzungen
Output-Disziplin	mittel	gering	durchgängig, schnell etablierbar
Memory-Funktionen	mittel	mittel	wiederkehrende Kontexte über Sitzungen
Periodischer Reset	mittel	gering	klar abgegrenzte Einzelaufgaben
RAG / Retrieval	hoch	hoch	große, stabile Wissensbestände

6 Handlungsempfehlungen für Unternehmen

Für mittelständische Organisationen, die KI-Assistenten produktiv einsetzen, empfiehlt sich ein gestuftes Vorgehen statt isolierter Einzelmaßnahmen:

Stufe 1 – Transparenz schaffen

- Verbrauch sichtbar machen: Token- bzw. Credit-Reporting aktivieren und Budgets bzw. Limits je Team festlegen
- Baseline messen, bevor optimiert wird. Ohne Ausgangswert lässt sich keine Einsparung belegen

Stufe 2 – Quick Wins umsetzen

- Modell-Routing einführen: Standard auf ein günstiges Modell, leistungsstarke Modelle gezielt für komplexe Aufgaben
- Prompt-Caching für stabile Systemkontexte aktivieren und Prefixes konsistent halten
- Output-Disziplin als Prompt-Standard etablieren: Format und Länge vorgeben

Stufe 3 – Strukturell verankern

- Kontextstrategie definieren: klare Regeln für Sitzungslänge, Reset-Punkte und Einsatz von Memory/Summarization
- Bei großen Wissensbeständen Retrieval (RAG) aufbauen, statt Dokumente vollständig in den Kontext zu laden
- Befähigung der Anwender: Tokenkosten sind heute auch eine Frage des Arbeitsverhaltens. Kurze Schulungen zahlen sich überproportional aus

Leitprinzip

Tokenkosten entstehen nicht im Einkauf, sondern in der Nutzung. Die wirksamste Kombination für die meisten Organisationen ist: günstiges Standardmodell + konsequentes Caching + disziplinierte Kontext- und Output-Verwaltung. **Damit lassen sich die in Abbildung 3 gezeigten Größenordnungen realisieren, ohne Einbußen bei der Ergebnisqualität.**

7 Fazit

Die Umstellung von GitHub Copilot auf eine token-basierte Abrechnung hat ein strukturelles Thema in die Breite getragen: Der Preis von KI-Assistenz koppelt sich zunehmend an den realen Verbrauch. Was wie eine reine Preisfrage aussieht, ist in Wirklichkeit eine Frage des Designs von Arbeitsabläufen.

Der zentrale Mechanismus ist der kumulative Token-Effekt: Weil in jeder Runde der gesamte Kontext erneut verarbeitet wird, ist das ungebremste Füllen eines einzelnen Kontextfensters der teuerste aller Wege. Die Modellrechnung in Abbildung 3 zeigt die Größenordnung: Verdichtung und bewusste Kontextführung senken den Verbrauch um rund 70 %, ohne den Nutzen zu schmälern.

Die gute Nachricht: Die größten Hebel sind weder teuer noch kompliziert. Bewusste Modellwahl, Prompt-Caching und Output-Disziplin liefern hohe Einsparungen bei geringem Aufwand und sollten den Anfang bilden. Aufwändigere Bausteine wie Retrieval folgen dort, wo die Datenlage es rechtfertigt. Entscheidend ist, das Thema nicht als einmalige Einkaufsentscheidung zu behandeln, sondern als fortlaufende Disziplin zu verankern, getragen von Transparenz, klaren Regeln und befähigten Anwendern.

Wer den Tokenverbrauch bewusst steuert, gewinnt doppelt: planbare Kosten und durch fokussierten Kontext oft auch bessere, schnellere Ergebnisse. Die token-basierte Abrechnung ist damit weniger eine Bedrohung als ein Anlass, den eigenen KI-Einsatz reifer und wirtschaftlicher aufzustellen.

Antwort auf die Leitfrage

Am wirksamsten senken vier Maßnahmen den Tokenverbrauch: bewusste **Modellwahl**, **Prompt-Caching**, **Output-Disziplin** und eine kontrollierte **Kontextstrategie**. Sie bestätigen die eingangs formulierte These. Der entscheidende Hebel ist der Kontext, nicht das einzelne Modell und verbinden hohe Wirkung mit geringem Aufwand. In den Arbeitsalltag überführen lassen sie sich am einfachsten über verbindliche, maschinenlesbare Regeln. Genau das leistet die Vorlage in **Anhang A**.

Anhang A Sofort einsetzbare Vorlage: Token-Disziplin per Instruktionsdatei

Die zentrale Empfehlung des Fazits, Tokenkosten über verbindliche im Alltag wirksame Regeln zu steuern, lässt sich unmittelbar umsetzen. Werkzeuge wie GitHub Copilot und agentische Assistenten lesen projektweite Instruktionsdateien und richten ihr Verhalten danach aus. Die folgende Vorlage übersetzt die Maßnahmen aus Kapitel 4 in genau solche Regeln. Sie entfaltet ihren Nutzen, sobald sie im Projekt hinterlegt ist, ohne weitere Konfiguration.

A.1 So setzen Sie die Vorlage ein

- **Schritt 1 – Ablegen:** Den folgenden Inhalt als Datei `.github/copilot-instructions.md` (für GitHub Copilot) oder als **AGENTS.md** im Projekt-Stammverzeichnis (tool-übergreifend) speichern.
- **Schritt 2 – Anpassen:** Modellnamen und projektspezifische Vorgaben ergänzen; die Regeln bei Bedarf an interne Konventionen angleichen.
- **Schritt 3 – Verankern:** Die Datei mit ins Repository einchecken, damit sie für alle Beteiligten und alle Sitzungen automatisch gilt. So wird Token-Disziplin zum Standard statt zur Einzelentscheidung.

A.2 Die Vorlage

Die Datei wird als eigenständiger Download bereitgestellt; nachfolgend der vollständige Inhalt:

```
# Projekt-Instruktionen für KI-Assistenten - Token-Disziplin
#
# Zweck: token-effiziente, kostenbewusste Zusammenarbeit.
# Ablage: .github/copilot-instructions.md (GitHub Copilot) oder
#        AGENTS.md im Projekt-Stammverzeichnis (tool-übergreifend).
# Sprache der Antworten: Deutsch, sofern nicht anders gewünscht.

## 1. Antwortverhalten (Output-Disziplin)
- Antworte knapp und direkt. Keine Wiederholung der Frage, keine
  Einleitungs- oder Schlussfloskeln.
- Bei Code-Änderungen: liefere gezielte Diffs bzw. nur die geänderten
  Stellen, nicht die ganze Datei erneut.
- Erkläre nur, wenn ausdrücklich gefragt. Sonst genügt die Lösung.
- Keine redundante Zusammenfassung am Ende der Antwort.

## 2. Kontext und Dateien (nur das Nötige lesen)
- Lies ausschließlich die für die Aufgabe relevanten Dateien oder
  Abschnitte. Lies nicht „zur Sicherheit“ das gesamte Repository.
- Frage nach, bevor du großflächig (viele Dateien) liest oder änderst.
- Verweise auf Datei und Zeilenbereich, statt ganze Dateien zu kopieren.
- Wiederhole bereits gelesenen, unveränderten Kontext nicht.

## 3. Modellwahl und Reasoning-Aufwand
- Standard ist ein leichtgewichtiges, günstiges Modell für einfache
  Aufgaben (Formatierung, kleine Edits, Fragen, Boilerplate).
- Ein leistungsstarkes Modell nur für komplexe, mehrstufige oder
  architektonische Aufgaben verwenden.
- Den Reasoning-Aufwand nur erhöhen, wenn die Aufgabe es erfordert.

## 4. Caching-freundlich arbeiten
- Stabile, wiederkehrende Anweisungen bleiben unverändert am Anfang
  (cache-fähiger, identischer Prefix).
- Vermeide unnötige Änderungen an wiederkehrenden Prompt-Bestandteilen.

## 5. Sitzungsdisziplin (Kontextstrategie)
- Eine Sitzung entspricht einer abgegrenzten Aufgabe. Für eine neue
  Aufgabe eine neue Sitzung starten, statt einen langen Chat zu füllen.
- Bei langen Sitzungen den Verlauf regelmäßig verdichten
  (Zwischenstand zusammenfassen, Irrelevantes verwerfen).
```

6. Umfang und Sicherheit

- Bestätige Umfang und Plan, bevor du autonome Mehrdatei-Läufe oder weitreichende Änderungen startest.
- Im Zweifel kurz nachfragen, statt teure Sackgassen-Läufe zu erzeugen.

A.3 Welcher Schritt welchen Mehrwert bringt

Jede Regel der Vorlage ist an eine Maßnahme aus diesem Whitepaper geknüpft und zahlt direkt auf die Tokenkosten ein:

Regel aus der Vorlage	Maßnahme (Kapitel)	Erwarteter Mehrwert
Knappe Antworten; Diffs statt ganzer Dateien	Output-Disziplin (4.6)	Weniger Output-Tokens, die teuerste Token-Art
Nur relevante Dateien/Abschnitte lesen	Retrieval statt Stuffing (4.5)	Weniger Input-Tokens je Anfrage
Leichtes Standardmodell, starkes Modell nur bei Bedarf	Modellwahl & Routing (4.4)	Bis zu ~20× geringerer Input-Preis je Token
Stabiler, unveränderter Prefix	Prompt-Caching (4.3)	Bis zu ~90 % geringere Input-Kosten bei Cache-Treffern
Neue Sitzung je Aufgabe; Verlauf verdichten	Kontextstrategie (4.1)	Vermeidet quadratisches Wachstum – ~70 % in der Modellrechnung
Umfang vor autonomen Läufen bestätigen	Sitzungs-/Scope-Disziplin (Kap. 6)	Verhindert teure Sackgassen-Läufe

Der Mehrwert ist somit unmittelbar: Wer die Vorlage hinterlegt, setzt die wirksamsten Empfehlungen dieses Whitepapers in einem einzigen, dauerhaften Schritt um und macht token-effizientes Arbeiten zum reproduzierbaren Standard.

Methodik, Grenzen und Quellen

Faktische Aussagen zur Copilot-Abrechnung, zu Modellpreisen, zu Caching-Größenordnungen sowie zur Tokenisierung stützen sich auf öffentlich zugängliche Anbieterdokumentation und Branchenberichterstattung (u. a. GitHub-Dokumentation und -Blog, TechCrunch, The New Stack, Provider-Dokumentation zu Prompt-Caching sowie Fachbeiträge und Studien zur Tokenisierung). Die in Abbildung 1, 4 und 5 dargestellten Werte geben gemeldete bzw. veröffentlichte Größen wieder; die Richtwerte im Worked Example in Abschnitt 2.2 sind gerundet und tokenizer-abhängig.

Abbildung 2 ist eine schematische Darstellung der Tokenisierung; die tatsächliche Zerlegung hängt vom jeweiligen Tokenizer ab. Abbildung 3 ist eine Modellrechnung auf Basis offengelegter Annahmen (System-Prefix 1.000, User 150, Assistant 350 Tokens je Turn; Re-Primer 800 Tokens; Summarization-Deckel 1.500 Tokens). Sie dient der Veranschaulichung des kumulativen Effekts und stellt keine Messung dar; die zugrunde liegende Token-Mathematik ist jedoch deterministisch. Abbildung 6 ist eine qualitative Einordnung als Entscheidungshilfe.

Grenzen der Analyse. Die genannten Größenordnungen sind keine garantierten Einsparungen. Der tatsächliche Effekt hängt vom konkreten Workload, vom eingesetzten Tokenizer, von der Cache-Trefferquote sowie von den jeweils gültigen Anbieterkonditionen ab, die sich ändern können. Die Modellrechnung abstrahiert zudem von Nebenkosten einzelner Maßnahmen (etwa den Output-Tokens einer Zusammenfassung). Die Aussagen sind daher als belastbare Tendenzen zu verstehen, die im eigenen Kontext zu validieren sind, idealerweise gegen eine zuvor gemessene Baseline (siehe Kapitel 6).

© 2026 twenty5ai – Sebastian Oczachowski

Dieses Whitepaper dient der Information und ersetzt keine individuelle Beratung. Modellnamen und Preise können sich ändern; maßgeblich ist die jeweils aktuelle Anbieterinformation.