

M08b - Tokenizing & Chunking

Stand: 02.2025

Die effiziente Textverarbeitung beruht auf drei zentralen Elementen: der Wahl des richtigen Tokenizers, der optimalen Chunk-Größe und einer passenden Chunking-Strategie. Diese Faktoren bilden die Basis für eine erfolgreiche Dokumentenanalyse in NLP-Anwendungen. Im Folgenden erfahren Sie, wie Sie diese Parameter systematisch an die Eigenschaften Ihres Dokuments und die Anforderungen der Anwendung (z. B. Fragebeantwortung, Zusammenfassung, Code-Verarbeitung) anpassen und optimieren können.

1 Tokenizer-, Chunking- & Strategiewahl

1.1 Dokumenttypen

Dokumenttyp	Empfohlener Tokenizer	Chunk-Größe (Tokens)	Überlappung (%)	Empfohlene Chunking-Strategie	Begründung
Lange Texte	SentencePiece oder BPE	512–1024	20–30%	Semantisches & embeddingbasiertes Chunking	Diese Tokenizer zerlegen den Text in kleinere, semantisch sinnvolle Einheiten. Größere Chunks helfen, den Kontext beizubehalten und logische Einheiten in dichten Texten zu bewahren.

Dokumenttyp	Empfohlener Tokenizer	Chunk-Größe (Tokens)	Überlappung (%)	Empfohlene Chunking-Strategie	Begründung
Mittel-lange Texte	WordPiece	256–512	10–20%	Semantisches Chunking	WordPiece verarbeitet gemischte Sprache gut. Semantisches Chunking fasst narrative und strukturierte Abschnitte optimal zusammen, ohne den Text zu stark zu fragmentieren.
Kurze Texte	Whitespace-/Symbol-basierte Tokenizer	50–200	0–5%	Rekursives Zeichen-Chucking (bei unklaren Grenzen)	Kurze, oft stark strukturierte Texte profitieren von kleinen Chunks. Rekursives Zeichen-Chucking kann helfen, bei fehlenden klaren Grenzen die Struktur zu wahren.

Dokumenttyp	Empfohlener Tokenizer	Chunk-Größe (Tokens)	Überlappung (%)	Empfohlene Chunking-Strategie	Begründung
Code & Technische Dokumente	Whitespace- oder benutzerdefinierte symbolbasierte Tokenizer (mit funktionsspezifischen Regeln)	Basierend auf logischen Blöcken (z. B. pro Funktion oder Absatz, ca. 256 Tokens)	Variabel (idealerweise minimale Überlappung oder blockgrenzenangepasst)	Agentisches Chunking (unter Einbeziehung logischer und syntaktischer Strukturen)	Die strukturelle Integrität ist entscheidend, um die Semantik des Codes zu erhalten. Agentisches Chunking berücksichtigt funktionale Zusammenhänge und stellt die Intaktheit der Blöcke sicher.

1.2 Anwendungsszenarien

Szenario	Ziel	Empfohlenes Chunking	Empfohlene Strategie	Begründung
Antworten auf Fragen	Exakte Extraktion relevanter Passagen	Moderat bis große Chunks (512 Tokens bei langen Texten) mit hoher Überlappung (30–50%)	Kombination aus semantischem und embeddingbasiertem Chunking	Hohe Überlappung stellt sicher, dass der Kontext zwischen den Chunks nicht verloren geht. Semantische Grenzen und embeddingbasierte Analysen erfassen relevante Abschnitte präzise.
Zusammenfassungen	Verdichtung des Inhalts bei Beibehaltung der Kernaussagen	Mittlere Chunks (256 Tokens) mit moderater Überlappung (10–20%)	Semantisches Chunking	Semantisches Chunking bewahrt komplette Sinnabschnitte, sodass die Kernaussagen klar extrahiert werden können, ohne den Kontext zu verlieren.
Informationsretrieval (RAG)	Effiziente Auffindbarkeit relevanter Abschnitte	Chunks von 256–512 Tokens mit moderater Überlappung (10–20%)	Embeddingbasiertes Chunking	Embeddingbasiertes Chunking gruppiert semantisch verwandte Inhalte. So werden relevante Informationen leichter auffindbar und retrieval-technisch optimal aufbereitet.

Szenario	Ziel	Empfohlenes Chunking	Empfohlene Strategie	Begründung
Named Entity Recognition (NER)	Identifikation wichtiger Entitäten (Namen, Daten usw.)	Chunks, die an Satzgrenzen ausgerichtet sind (ca. 256 Tokens) und minimale Überlappung (5–15%)	Semantisches Chunking (ggf. kombiniert mit embeddingbasierten Ansätzen)	Durch an Satzgrenzen ausgerichtete Chunks wird vermieden, dass Entitäten aufgespalten werden. Eine embeddingbasierte Analyse kann zusätzlich helfen, zusammengehörige Entitäten zu erfassen.
Textklassifikation	Zuweisung von Labels zu Dokumenten oder Abschnitten	Größere, grobere Chunks (gesamtes Dokument oder 512 Tokens) mit wenig bis keiner Überlappung	Semantisches Chunking (optional mit reduzierter Granularität)	Größere Unterteilungen verhindern Rauschen, während semantische Einheiten erhalten bleiben, die für die Klassifikation relevant sind.
Code-Kommentierung/Erklärung	Verständnis und Erklärung von Codeabschnitten	Chunks, die durch logische Blöcke definiert sind (z. B. pro Funktion, Modul) mit Überlappung nur, wenn notwendig (blockgrenzenbezogen)	Agentisches Chunking	Agentisches Chunking berücksichtigt syntaktische und semantische Aspekte des Codes. So bleiben logische Zusammenhänge, wie Funktionsdefinitionen, erhalten und können optimal erklärt werden.



Tip

Bevor Sie eine konkrete Implementierung starten, sollten Sie Ihre Dokumente genau analysieren, um die für Ihren Anwendungsfall optimale Kombination aus Tokenizer, Chunk-Größe, Überlappung und Chunking-Strategie auszuwählen. Eine Pilotphase mit verschiedenen Einstellungen kann helfen, den besten Ansatz zu ermitteln.

2 Beispiel

```

# Original Text
text = "Maschinelles Lernen ist ein spannendes Thema."

# Schritt 1: Text zu Token
text_tokens = ["Masch", "inelles", "_Lernen", "_ist", "_ein", "_spannendes", "_Thema", "."]

# Schritt 2: Token zu IDs
token_ids = [2847, 1123, 892, 345, 287, 4561, 1876, 13]

# Schritt 3: Chunking (Chunk-Größe = 4)
chunks = [
    # Chunk 1: ["Masch", "inelles", "_Lernen", "_ist"]
    [2847, 1123, 892, 345],

    # Chunk 2: ["_ist", "_ein", "_spannendes", "_Thema"]
    [345, 287, 4561, 1876]
]

```

- Tokenizing:
 - Zerlegt Text in kleinste Einheiten (Token)
 - Diese Token werden in Zahlen (IDs) umgewandelt
 - Ein Token kann ein Wort, Teil eines Wortes oder ein Satzzeichen sein
- Chunking:
 - Gruppiert die Token in verarbeitbare Blöcke
 - Beispiel: Bei max. 4096 Token pro Anfrage werden längere Texte in Chunks aufgeteilt
 - Jeder Chunk behält dabei genug Überlappung (hier 1) zum vorherigen Chunk für Kontexterhalt
- Zusammenspiel:

- Text wird erst tokenisiert (in kleinste Einheiten zerlegt)
- Die Token werden dann in Chunks gruppiert (für Verarbeitung)
- Chunks werden nacheinander verarbeitet
- LLM behält Kontext zwischen Chunks durch Überlappungen

3 Parameter- und Strategieauswahl

- **Tokenizer-Auswahl:**
 - **SentencePiece/BPE** sind ideal für lange, unstrukturierte Texte, da sie feine Subworteinheiten erzeugen und dabei semantische Bedeutung beibehalten.
 - **WordPiece** ist optimal für hybride Texte, in denen technische sowie allgemeine Sprache vorkommen.
 - **Whitespace-/Symbol-basierte Tokenizer** (oder speziell angepasste Tokenizer für Code) gewährleisten, dass die Struktur, beispielsweise in kurzen Texten oder Quellcode, erhalten bleibt.
- **Chunk-Größe und Überlappung:**
 - Die **Chunk-Größe** wird so gewählt, dass jeweils eine komplette logische Einheit erfasst wird. Längere Texte benötigen größere Chunks, während bei kurzen Texten kleinere, präzisere Segmente ausreichend sind.
 - **Überlappung** hilft dabei, Kontextinformationen am Rand der Chunks nicht zu verlieren. Für komplexe Aufgaben (wie präzise Fragebeantwortung) ist eine höhere Überlappung vorteilhaft, wohingegen bei Aufgaben wie Klassifikation geringere Überlappungen ausreichend sind.
- **Zusätzliche Chunking-Strategien:**
 - **Semantisches Chunking** zielt darauf ab, thematisch und inhaltlich zusammenhängende Abschnitte zu bilden.
 - **Rekursives Zeichen-Chucking** eignet sich, wenn keine klaren sprachlichen Grenzen vorliegen oder bei sehr strukturierten, kurzen Dokumenten.
 - **Embeddingbasiertes Chunking** nutzt Ähnlichkeiten im Einbettungsraum, um semantisch verwandte Inhalte zu gruppieren, was insbesondere bei Retrieval-Aufgaben nützlich ist.
 - **Agentisches Chunking** verwendet agentenbasierte Verfahren, um logische und syntaktische Zusammenhänge zu identifizieren – ein Ansatz, der besonders bei Code und technischen Dokumenten Vorteile bietet.
- **Praktische Rahmenbedingungen:**

- **Speicherverbrauch und Verarbeitungsgeschwindigkeit** lassen sich durch Anpassung der Chunk-Größe steuern. Kleinere Chunks reduzieren den Speicherbedarf und beschleunigen die Verarbeitung, was vor allem bei großen Datenmengen von Bedeutung ist.
- **Kosten** können durch die Optimierung der Überlappung minimiert werden. Eine zu hohe Überlappung erhöht Redundanzen und Rechenaufwand, sodass hier ein ausgewogenes Verhältnis gefunden werden muss.

Tip:

Erstellen Sie eine Checkliste für die Parameterwahl, die alle wesentlichen Aspekte – von Tokenizer-Auswahl über Chunk-Größe bis hin zur konkreten Chunking-Strategie – abdeckt. Dies unterstützt Sie dabei, systematisch vorzugehen und sicherzustellen, dass alle praktischen Rahmenbedingungen berücksichtigt werden.

4 Optimierung für verschiedene Modellgrößen

- **Große Modelle:**
 - Können größere Chunk-Größen (bis zu 1024 Tokens) verarbeiten, wodurch mehr Kontext erhalten bleibt.
 - Profitieren in Szenarien wie der Fragebeantwortung von einer höheren Überlappung (30–50%) und einer Kombination aus semantischem und embeddingbasiertem Chunking.
- **Kleinere Modelle:**
 - Sollten kleinere Chunk-Größen (z. B. 256–512 Tokens) verwenden, um den Speicherbedarf und die Verarbeitungsgeschwindigkeit zu optimieren.
 - Eine geringere Überlappung (10–20%) ist empfehlenswert, um unnötige Redundanz zu vermeiden, während dennoch ausreichend Kontext für die jeweilige Aufgabe erhalten bleibt.
- **Iterative Feinabstimmung:**
 - Es empfiehlt sich, anhand von Metriken wie F1-Score (bei Fragebeantwortung), ROUGE (bei Zusammenfassungen) und Recall@K (bei Retrieval-Aufgaben) verschiedene Einstellungen zu evaluieren und schrittweise zu optimieren.
 - Szenariospezifische Experimente können helfen, die Wahl des Tokenizers, die Chunk-Größe, die Überlappung und die jeweils verwendete Chunking-Strategie iterativ anzupassen.

**Tip**

Nutzen Sie automatisierte Tests und Monitoring-Tools, um die Leistung Ihrer Modelle kontinuierlich zu überwachen und dynamisch Anpassungen an den Chunking-Parametern vorzunehmen. Eine regelmäßige Evaluation ermöglicht es, auf Veränderungen im Input oder in den Anforderungen schnell zu reagieren.

5 Anhang: Checkliste Parameterwahl

- **1. Analyse der Dokumenteigenschaften**

- ☐ **Dokumenttyp identifizieren:** Lange Texte, mittel-lange Texte, kurze Texte, Code/technische Dokumente.
- ☐ **Typische Eigenschaften erfassen:** Länge, Struktur, Informationsdichte, spezielle Formatierungen (z. B. Tabellen, Codeblöcke).

- **2. Auswahl des Tokenizers**

- ☐ **Sprachliche und technische Anforderungen prüfen:**
 - Lange, unstrukturierte Texte: SentencePiece oder BPE
 - Gemischte Inhalte (technisch und allgemein): WordPiece
 - Stark strukturierte Daten oder Code: Whitespace-/Symbol-basierte Tokenizer oder angepasste Tokenizer
- ☐ **Spezifische Anforderungen an Subwort-Einheiten bewerten.**

- **3. Festlegung der Chunk-Größe**

- ☐ **Ziel der Chunking-Einheit definieren:** Soll ein vollständiger Satz, Absatz oder logische Einheit abgebildet werden?
- ☐ **Dokumenttyp berücksichtigen:**
 - Lange Texte: 512–1024 Tokens
 - Mittel-lange Texte: 256–512 Tokens
 - Kurze Texte: 50–200 Tokens
 - Code/technische Dokumente: Abhängig von logischen Blöcken (z. B. pro Funktion)
- ☐ **Praktische Rahmenbedingungen einbeziehen:** Speicherverbrauch und Verarbeitungsgeschwindigkeit.

- **4. Definition der Überlappung**

- ☐ **Ziel des Überlappungsgrades festlegen:** Sicherstellung des Kontext-Erhalts, ohne unnötige Redundanz.
- ☐ **Empfohlene Überlappungswerte anpassen:**
 - Hohe Überlappung (30–50%) für kontext-sensitive Aufgaben wie Q&A.
 - Geringere Überlappung (0–5% bis 10–20%) für Klassifikation oder strukturierte Daten.

- **5. Auswahl der konkreten Chunking-Strategie**

- ☐ **Strategie zur Erhaltung semantischer Einheiten prüfen:**

- Semantisches Chunking, um zusammenhängende inhaltliche Blöcke zu bilden.
- ☐ **Alternativen in Betracht ziehen, falls keine klaren Grenzen vorliegen:**
 - Rekursives Zeichen-Chucking.
- ☐ **Spezialfälle für Retrieval und NER:**
 - Embeddingbasiertes Chunking, um semantisch verwandte Abschnitte zu gruppieren.
- ☐ **Besondere Anforderungen bei Code:**
 - Agentisches Chunking, um logische und syntaktische Zusammenhänge zu berücksichtigen.
- **6. Evaluation und iterative Feinabstimmung**
 - ☐ **Metriken definieren:** F1-Score, ROUGE, Recall@K usw.
 - ☐ **Pilotphase durchführen:** Verschiedene Einstellungen testen und Ergebnisse vergleichen.
 - ☐ **Parameter anpassen:** Basierend auf den Evaluierungsergebnissen systematisch justieren.
- **7. Monitoring und praktische Rahmenbedingungen**
 - ☐ **Automatisierte Tests und Monitoring einrichten:** Zur kontinuierlichen Überwachung der Modellleistung.
 - ☐ **Kosten und Ressourcenverbrauch im Blick behalten:** Speicher, Rechenleistung und Kosten optimieren.
 - ☐ **Regelmäßige Überprüfung:** Auf Veränderungen im Input oder in den Anforderungen reagieren und Parameter entsprechend anpassen.

Diese Checkliste unterstützt Sie dabei, einen strukturierten Ansatz zu verfolgen, sodass alle relevanten Parameter und praktischen Rahmenbedingungen systematisch berücksichtigt werden.