HSLU

AI/ML

Effiziente Text-Generierung mit spekulativem Decoding

Bachelorthesis

Autor: Flavio Kluser

Advisor Dr. Andreas Marfurt (andreas.marfurt@hslu.ch)
Hochschule Luzern

Experte Ahmed Harbaoui (ahmed@paretolabs.ch)
ParetoLabs

Hochschule Luzern - Departement Informatik
Artificial Intelligence & Machine Learning

Rotkreuz, 6. Juni 2024

Bachelorarbeit an der Hochschule Luzern – Informatik

Titel: Effiziente Text-Generierung mit spekulativem Decoding

Student: Flavio Kluser

Studiengang: B.Sc. Artificial Intelligence and Machine Learning

Abschlussjahr: 2024

Betreuungsperson: Dr. Andreas Marfurt

Expertin/Experte: Ahmed Harbaoui

Auftraggeberin/Auftraggeber: Dr. Andreas Marfurt

Codierung / Klassifizierung der Arbeit:

- ☒ Öffentlich (Normalfall)
- ☐ Vertraulich

Eidesstattliche Erklärung
Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig und ohne unerlaubte fremde Hilfe angefertigt habe, alle verwendeten Quellen, Literatur und andere Hilfsmittel angegeben habe, wörtlich oder inhaltlich entnommene Stellen als solche kenntlich gemacht habe das Vertraulichkeitsinteresse des Auftraggebers wahren und die Urheberrechtsbestimmungen der Hochschule Luzern respektieren werde.

Rotkreuz, 6. Juni 2024 _____

Abgabe der Arbeit auf der Portfolio Datenbank:
Bestätigungsvisum Studentin/Student
Ich bestätige, dass ich die Bachelorarbeit korrekt gemäss Merkblatt auf der Portfolio Datenbank ablege. Die Verantwortlichkeit sowie die Berechtigungen gebe ich ab, so dass ich keine Änderungen mehr vornehmen oder weitere Dateien hochladen kann.

Rotkreuz, 6. Juni 2024 _____

Verdankung

Ich bedanke mich herzlich bei meinem Betreuer Dr. Andreas Marfurt für die durchgehende Unterstützung dieser Bachelorarbeit sowie die fachliche Expertise. Die wöchentlichen Meetings haben mich motiviert und sehr wesentlich zum Fortschritt und Erfolg dieser Arbeit beigetragen.

Ebenfalls bedanke ich mich bei Andreas Waldis für die Bereitstellung von Rechenleistung vom Forschungshub.

Abstract

Im Rahmen dieser Bachelorarbeit werden Ansätze für die Verbesserung von spekulativem Decoding untersucht. Spekulatives Decoding ermöglicht eine schnellere und effizientere Generierung bei Large Language Models (LLMs) mit dem Einsatz eines kleineren Draft-Modells. Durch die Generierung von mehreren Tokens mit dem Draft-Modell und die anschliessende Validation mit dem Target-Modell in einem einzigen Forward-Pass kann die Generierung erheblich beschleunigt werden. Diese Arbeit untersucht, ob spekulatives Decoding mit einem entropiebasierten Ansatz weiter verbessert werden kann. Dieser Ansatz ermöglicht es, Tokens zu generieren, bis das Draft-Modell unsicher wird. Es konnte nachgewiesen werden, dass die Entropie ein guter Prädiktor für die Akzeptanzwahrscheinlichkeit ist. Durch einen zweistufigen Aufbau aus Simulation und Implementation auf Basis der Hugging Face Transformers Bibliothek, konnte der neue Ansatz Entropy Drafting umfassend getestet und evaluiert werden. Dabei wurden drei Strategien entwickelt und als Abbruchkriterium für die Drafting-Phasen ausgewertet. Durch die Auswertungen wird nachgewiesen, dass Entropy Drafting nicht nur in der Theorie funktioniert, sondern auch praktisch messbare Kostenvorteile mit sich bringt. Die Generierung hat sich im Vergleich mit der bisherigen Hugging Face Implementation von spekulativem Decoding bis zu 1.07x beschleunigt. Im Vergleich zu Generierung ohne Assistenzmodell wurde bis zu 1.89x schneller generiert. Weitere Experimente mit grösseren Modellen und längeren Ausgaben zeigen keine Verbesserung der Ergebnisse. Die Arbeit liefert aufschlussreiche Erkenntnisse über die besten Strategien und Hyperparameter für Entropy Drafting. Zusammen mit der Implementation und den entwickelten Tools ebnen diese den Weg für zukünftige Forschung an diesem Ansatz.

Inhaltsverzeichnis

1. Einleitung

1.1. Motivation

Large Language Models (LLMs) haben in den vergangenen Jahren stark an Bedeutung gewonnen. Allein das LLM ChatGPT von OpenAI hat innert 5 Tagen eine Million User erreicht (Deng et al., 2023). Als unterliegende Technologie von kommerziellen Services wie ChatGPT, Google Bard oder Bing Chat ermöglichen LLMs sprachbezogene Anwendungen wie Chatbots, Sentiment-Analyse, Zusammenfassung, Frage/Antwort, etc. Durch die Einführung von multimodalen LLMs wird das Spektrum der Anwendungsmöglichkeiten nochmals erheblich erweitert, da dieser Typ von LLM mit Bild- und Audiodateien umgehen kann.

Als Technologie, die starkes Wachstum und zunehmende Adaption in Industrie geniesst, sind Optimierungsmöglichkeiten ein gefragtes Gebiet der Forschung. Ein Kostenmodell von SemiAnalysis schätzt die täglichen Betriebskosten von ChatGPT im Februar 2023 auf 694'444 $ (Patel & Afzal, 2023). Durch das starke Wachstum von OpenAI und der Einführung des wesentlichen rechenintensiveren GPT-4 Modells ist davon auszugehen, dass die Betriebskosten weiterhin gestiegen sind. Aus finanzieller Sicht ist eine Optimierung von LLMs in Hinsicht auf die beanspruchten Rechenressourcen für Betreiber in jedem Fall erstrebenswert.

Ein weiterer Aspekt ist die Geschwindigkeit. Auto-Regressive Large Language Models (AR-LLM) verwenden zur Generierung eines neuen Tokens die vorhergehenden Tokens als Input. Die sequenzielle Generierung hat zur Folge, für jeden Token ein Forward-Pass durch das Modell notwendig ist. Dies kann insbesondere bei grossen Sprachmodellen zu einer erheblichen Latenz bei der Generierung von Text führen (C. Chen et al., 2023).

1.2. Zielsetzung

Das Ziel dieser Arbeit ist es, den Ansatz *Speculative Decoding* mit der Implementation einer Methode zu verbessern und systematisch zu evaluieren. Dafür werden in der Aufgabenstellung A.5 zwei Methoden vorgeschlagen.

- Entropie: Adaptive Anzahl an Tokens basierend auf der Decoding-Entropie generieren
- Mehrere Kandidaten: Mehrere Kandidaten (z.B. mit Beam Search) generieren, um die Übereinstimmung mit dem Target-Modell zu erhöhen

In Absprache mit der Betreuungsperson wurde entschieden, den Ansatz Entropie zu verfolgen. Gemäss Aufgabenstellung werden Experimente mit Open-Source LLMs durchgeführt. Zusätzlich können auch Quantisierung oder eine Verteilung auf mehrere GPUs realisiert werden, um mit sehr grossen Modellen zu arbeiten.

1.3. Abgrenzung

Diese Arbeit konzentriert sich auf Speculative Decoding und behandelt Knowledge Distillation (KD) nicht. Bei KD lernt ein kleines Studentenmodell von einem grossen Lehrermodell.

Dadurch kann das Wissen eines grossen Modells durch die Destillation stark komprimiert werden. KD setzt auf eine Teacher-Student Architektur (Gou et al., 2021). Dies weist gewisse Parallelen mit Speculative Decoding auf, da hier ein Target- und Draft-Modell verwendet werden. Diese werden z.T. Lehrer- und Studentenmodell genannt, was zu Verwechslungen führen kann. Speculative Decoding ist ein reiner Inferenzprozess, es werden keine Modelle trainiert.

Es werden nur bestehende Open-Source LLMs für diese Arbeit verwendet. Ein wichtiges Kriterium hierfür ist, dass das Modell in geeigneten Parametergrössen für den Einsatz als Target- und Draft-Modell angeboten wird. Es werden keine eigenen Modelle trainiert oder fine-tuned.

2. Stand der Forschung

2.1. Problemstellung

Eine fundamentale Limitation von AR-LLMs ist die autoregressive Generierung in Inferenz. Während es möglich ist, diese Modelle in Batches parallelisiert zu trainieren, funktioniert die Inferenz autoregressiv. Das Sampling eines neuen Wortes in einem Satz ist abhängig von den vergangenen Wörtern.

$$p(w_{t+1}|w_{\leq t})$$

Als praktischen Beispiel bedeutet das, dass ein übersetzter Text Wort-für-Wort generiert wird. Die Decoding-Schritte nutzen die Hardware nicht optimal aus. Für die meisten Anwendungen ist autoregressives Sampling stark von der Speicherbandbreite abhängig und kann nicht von moderner Beschleunigungshardware profitieren (Shazeer, 2019). Leviathan et al. (2023) stellen fest, dass Inferenz meist nicht durch Rechenkapazität (d. h. die arithmetischen Operationen) limitiert wird, sondern von der Speicherbandbreite und Kommunikation. Die Diskrepanz zwischen arithmetischen Operationen und Speicherbandbreite ist auch als *Memory Wall* Problem bekannt. Dies ist besonders relevant bei grossen, verteilten LLM Settings und Single-Batch Inferenz (Kim et al., 2024).

2.2. Decoding Strategien

In diesem Themenbereich spielen LLM Decoding Strategien eine zentrale Rolle. Um Mehrdeutigkeiten zu verhindern und präzise Begriffe im Rahmen dieser Arbeit zu etablieren, lohnt es sich, Decoding Strategien zu resümieren. Die Decoding-Strategien teilen sich in Deterministische Methoden und Stochastische Methoden.

2.2.1. Deterministische Methoden

Deterministische Methoden führen immer zum selben Resultat, solange sich Verteilungen und eventuelle Decoding Hyperparameter nicht ändern. Es gibt also kein Zufallsfaktor bei der Generierung (Gong & Yao, 2023). Die Methoden Greedy Search und Beam Search werden kurz zusammengefasst.

Greedy Search

Greedy Search wählt in jedem Schritt das Token mit der höchsten Wahrscheinlichkeit. Es ist die einfachste Decoding-Strategie. Ein Nachteil von Greedy Search ist, dass in jedem Schritt eine lokale Maximierung angestrebt wird, während der Gesamterfolg über mehrere Schritte nicht berücksichtigt wird. Darum kann Greedy Search in lokalen Optima hängen bleiben (Shi et al., 2024).

Beam Search

Beam Search behält bei jedem Schritt einen *Beam* mit den k wahrscheinlichsten Sequenzen. Mit dem Hyperparameter k kann die Breite des Beams (*Beam Width*) verändert werden.

Mit einem hohen k können mehr Sequenzen erkundet werden, was den Berechnungsaufwand erhöht (Shi et al., 2024). Da Beam Search kein Zufallsfaktor enthält, gehört der Ansatz zu den deterministischen Strategien.

Deterministische Methoden eignen sich grundsätzlich besser für *closed-ended* Tasks wie Übersetzung und Zusammenfassung als stochastische Methoden. Sie tendieren dazu, Instruktionen besser zu befolgen und weniger zu halluzinieren. Für *open-ended* Tasks sind deterministische Methoden nicht geeignet, da sie anfällig auf Textrepetitionen sind und durch den fehlenden Zufallsfaktor vorhersehbar und unkreativ generieren (Shi et al., 2024).

2.2.2. Stochastische Methoden

Bei stochastischen Methoden werden Tokens durch Sampling gewonnen. Das Sampling führt einen Zufallsfaktor ein. Dadurch sind diese Methoden nicht mehr deterministisch und die Resultate variieren selbst bei denselben Verteilungen und Decoding Hyperparametern (Gong & Yao, 2023). Die Methoden Temperature Sampling, Top-p Sampling und Top-k Sampling werden kurz zusammengefasst.

Temperature Sampling

Mit Temperature Sampling wird die Wahrscheinlichkeitsverteilung mit dem Hyperparameter τ angepasst. Mit der Temperatur τ wird die Verteilung schärfer oder flacher eingestellt. Für $\tau < 1$ wird die Verteilung schärfer, indem die Diskrepanz zwischen wahrscheinlichen und unwahrscheinlichen Tokens vergrössert wird. Für $\tau > 1$ wird die Verteilung flacher, was unwahrscheinliche Tokens begünstigt (Shi et al., 2024). Für $\tau = 0$ wird die Diskrepanz zwischen den Wahrscheinlichkeiten so gross, dass der Algorithmus greedy wird (Chang et al., 2023).

$$q_i = \frac{exp(z_i/\tau)}{\sum_j exp(z_j/\tau)}$$

Die Wahrscheinlichkeit q_i für ein Logit z_i ergibt sich durch einen Vergleich mit allen anderen Logits j.

Top-p Sampling

Top-p Sampling (auch bekannt als Nucleus Sampling) berücksichtigt die wahrscheinlichsten Tokens, bis die kumulative Wahrscheinlichkeit p erreicht oder übersteigt wird (Holtzman et al., 2020). p ist ein Hyperparameter.

$$\sum_{x \in V^{(p)}} P(x|x_{1:i-1}) \geq p$$

Wie in Holtzman et al. (2020) beschrieben wird ein Subset $V^{(p)}$ aus Tokens definiert. Dieses ist das kleinste Set, das die obenstehende Bedingung erfüllt.

Top-k Sampling

Top-k Sampling wählt als Subset für das Sampling die k wahrscheinlichsten Tokens aus der Verteilung. Dies führt dazu, dass nur wahrscheinlichen Tokens in das Sampling einfliessen (Shi et al., 2024). k ist ein Hyperparameter.

Stochastische Methoden eignen sich vor allem für *open-ended* Tasks, die von Kreativität profitieren. Durch den Zufallsfaktor sind sie anfälliger für Halluzinationen. Die für Sampling

benötigte Rechenleistung ist unerheblich, es bestehen keine Performancenachteile zu deterministischen Methoden (Shi et al., 2024).

2.3. Bestehende Forschung

Dieses Kapitel ist das Resultat der Literaturrecherche und konzentriert sich auf die wichtigsten bestehenden Arbeiten in diesem Feld. Es wurden auch Publikationen einbezogen, die andere Ansätze als Speculative Decoding verwenden.

2.3.1. Blockwise Parallel Decoding

Stern et al. (2018) haben einen Ansatz vorgestellt, mit dem die Generierung verlustfrei beschleunigt werden kann. Dafür wird die Architektur des Transformers um ein Multi-Output Feedforward Layer nach dem originalen Decoder Output Layer ergänzt. Diese Änderung ermöglicht es, zusätzlich zum nächsten Token p_1 auch eine beliebige Anzahl an zusätzlichen Tokens $p_2, ..., p_k$ vorauszusagen. Anschliessend werden alle Tokens parallel validiert. Um zu bestimmen, wie viele Tokens akzeptiert werden, werden Top-k Selection und Distance-Based Selection vorgeschlagen. Im Vergleich zu einem Baseline Greedy Decoder werden Beschleunigungen von bis zu 2x beobachtet, mit leichtem Qualitätsverlust sogar bis zu 7x (Stern et al., 2018). Nachteile von diesem Ansatz sind, dass zusätzliches Training auf dem angepassten Modell notwendig sind und nur Greedy Decoding unterstützt wird (Leviathan et al., 2023).

2.3.2. Speculative Decoding

Der in Leviathan et al. (2023) beschriebene Ansatz *Speculative Decoding* beschleunigt die Inferenz eines Modells und erfordert keine Änderungen an der Architektur oder des Trainings. Somit ist der Ansatz sehr gut auf bestehende Modelle generalisierbar. Die Idee wurde von *Speculative Execution* inspiriert. Speculative Execution ist eine bei Prozessoren verbreitete Optimierungstechnologie. Eine Operation wird parallel ausgeführt, um zu eruieren, ob sie überhaupt benötigt wird. Dies führt zu einem gewissen Grad an Parallelisierung und vermindert Verzögerungen (Burton, 1985).

Übertragen auf Speculative Decoding wird die Kernidee von Speculative Execution durch den Einsatz von zwei Modellen realisiert. Ein grosses aber langsames *Target-Model* und ein kleineres aber schnelleres *Draft-Model*. Das Draft-Modell generiert γ neue Tokens. Das Target Modell evaluiert mit einem einzigen Forward-Pass alle γ Vorschläge und generiert zudem den nachfolgenden Token. Im besten Fall werden somit $\gamma + 1$ Tokens generiert. Im schlechtesten Fall wird dank Speculative Sampling immerhin 1 Token generiert.

Die Autoren haben in Experimenten mit dem Modell T5 (Raffel et al., 2020) Laufzeitverbesserungen im Rahmen von 2x bis 3x gemessen. Als Tasks wurden Übersetzung und Zusammenfassung ausgewählt (Leviathan et al., 2023). Die Methode erfordert durch den Einsatz eines Assistenzmodells mehr Rechenoperationen. Vorteile dieser Methode sind, dass weder Änderungen an der Architektur noch Retraining erforderlich sind.

Fast parallel wurde von DeepMind ein Paper veröffentlicht, dessen Kernideen besonders beim Speculative Decoding sehr ähnlich sind. Der Fokus liegt hier aber auf einem verteilten Rechensetting für grosse Modelle. Für die Experimente wurden 16 Tensor Processing Units (TPUs) verwendet. Zudem wurde das Speculative Sampling hier mit Rejection Sampling realisiert (C. Chen et al., 2023).

Zudem schlägt das Paper eine neue Sampling-Methode vor, *Speculative Sampling*.

Speculative Sampling

Es wird angenommen, dass q die Draft-Verteilung, p die Target-Verteilung und x ein Token ist.
Dieses Sampling Verfahren akzeptiert Tokens, wenn $q(x) \leq p(x)$. In diesem Fall ist die Target-Wahrscheinlichkeit grösser als die Draft-Wahrscheinlichkeit. Falls $q(x) > p(x)$ wird das Sample mit der Wahrscheinlichkeit $1 - \frac{p(x)}{q(x)}$ verworfen und ein neues Sample von der justierten Verteilung $p'(x) = norm(max(0, p(x) - q(x))$ gezogen. Es wird bewiesen, dass die justierte Verteilung $p'(x)$ identisch mit $p(x)$ ist. (Leviathan et al., 2023).

Dadurch ist es möglich, das erste abgelehnte Token von einer justierten Verteilung $p'(x)$ neu zu samplen. Das erspart einen Forward-Pass, um $p(x)$ zu ermitteln.

2.3.3. Speculative Decoding with Big Little Decoder

In Kim et al. (2023) wird Framework vorgestellt, das auf Speculative Decoding aufbaut. Wie in Speculative Decoding wird ein kleines Modell zur Generierung von Tokens eingesetzt, zusammen mit einem grossen Modell für die Validierung der Tokens. Das Paper beschreibt zwei Policies zur Koordination. Eine Fallback Policy entscheidet, wann das Draft-Modell die Generierung stoppen und den Kandidaten an das Target-Modell geben soll. Die Fallback Policy entscheidet anhand der maximalen Wahrscheinlichkeit einer Token-Voraussage. Unterschreitet diese einen Threshold, wird ein Fallback zum Target-Modell ausgelöst. Zudem gibt es eine Rollback Policy. Durch eine Distanzmetrik werden die Kandidaten des Draft-Modells mit den Voraussagen des Target-Modells verglichen. Wird der Threshold unterschritten, werden alle Voraussagen bis zur betroffenen Position verworfen (Kim et al., 2023).

2.3.4. Lookahead Decoding

In Fu et al. (2024) wird *Lookahead Decoding* vorgestellt, ein neuartiger Ansatz zur Beschleunigung von Inferenz. Anders als bei Speculative Decoding und MEDUSA funktioniert Lookahead Decoding nicht mit einem 'guess and verify' Schema. Es werden kein Draft-Modell oder zusätzliche Decoding-Heads benötigt. In Lookahead Decoding werden vergangene N-Gramme in einem Cache gespeichert. Diese ermöglichen die Voraussage von N-Grammen. Es können mehrere Tokens parallel dekodiert werden, indem das Decoding als nicht-lineares Gleichungssystem behandelt und mit dem Jacobi-Verfahren gelöst wird. Die Hauptlimitation von Lookahead Decoding liegt in den zusätzlich nötigen Rechenoperationen für das parallele Decoding (Fu et al., 2024).

2.3.5. MEDUSA

Nicht alle Ansätze setzen auf ein Draft Modell. In MEDUSA werden durch das Anfügen von Decoding-Heads mehrere Fortsetzungen bestehend aus mehreren Tokens parallel generiert. Abbildung 2.1 zeigt eine vereinfachte Architektur von MEDUSA. Die im Paper beschriebenen Methoden ermöglichen es, das Fine-Tuning für die zusätzlichen Heads entweder eigenständig (Frozen Backbone) oder zusammen mit dem Modell (Joint Training) zu trainieren. Bestehende Modelle können problemlos damit erweitert werden. Das Framework MEDUSA umfasst weiter *self-distillation*. Self-Distillation ist eine Technik, die das Training der Heads ermöglicht, selbst wenn das ursprüngliche Trainings-Dataset für das Modell nicht zur Verfügung steht. Weiter wird *typical acceptance scheme* eingeführt. Dies ist ein Sampling Ansatz, der im Gegensatz zu Rejection Sampling plausible Kandidaten auch anhand ihrer Entropie bestimmt (Cai et al., 2024).

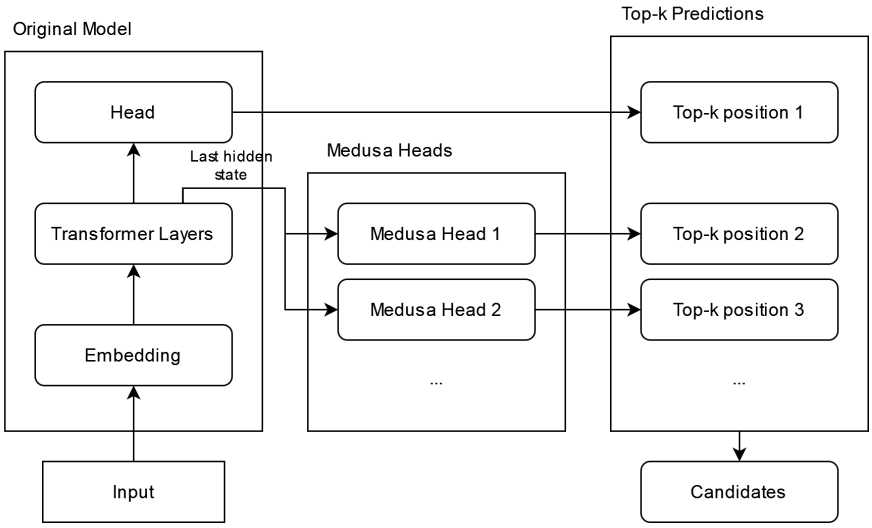


Abbildung 2.1.: Vereinfachte Darstellung von MEDUSA

2.3.6. Fazit

Im letzten Jahr wurden verschiedene Ansätze zur Beschleunigung der Generierung von LLMs veröffentlicht. Die Ideen sind dabei sehr unterschiedlich und ziehen Vor- und Nachteile mit sich. Speculative Decoding funktioniert ohne Fine-Tuning oder Änderungen an der Modellarchitektur. Allerdings funktioniert der Ansatz nur mit einem Assistenzmodell. MEDUSA erübrigt ein Assistenzmodell, erfordert aber das Fine-Tuning von zusätzlichen Decoding-Heads. Lookahead Decoding benötigt weder Assistenzmodell noch Fine-Tuning. Allerdings werden zusätzliche Rechenoperationen für die Lookahead Strategie benötigt.

Durch die Aufgabenstellung ist schon gegeben, dass der Ansatz auf Speculative Decoding aufbaut. Dennoch lohnt sich ein Blick in Publikationen für andere Ansätze. Dort werden einerseits die Schwächen von Speculative Decoding thematisiert, andererseits interessante neue Ansätze beschrieben. Beides ist für diese Arbeit relevant.

2.4. Aktuelle Implementation in Transformers

Als Codebasis für diese Arbeit wird die Library Transformers von Hugging Face verwendet. Die Implementation baut auf Assisted Generation (Gante, 2023) auf. Assisted Generation hat dieselbe Kernidee wie Speculative Decoding: Das autoregressive Generieren von mehreren Tokens mit dem Assistenzmodell, die anschliessend durch einen einzigen Forward-Pass validiert werden. Assisted Generation nutzt eine leicht andere Terminologie und nennt das Draft-Modell Assistenzmodell. Beide Begriffe sind bedeutungsgleich.

Assisted Generation wurde initial mit der Version v4.29.0 (veröffentlicht am 10. Mai 2023) in die Transformers Library aufgenommen. Unterstützt werden Greedy Decoding und Sampling. Letzteres kann durch do_sample=True aktiviert werden und benutzt multinominales Sampling (LysandreJik, 2023). Für Anwendungen, die stark Input-basiert sind (z.B. Übersetzung), wird die Verwendung von Greedy Decoding empfohlen. Bei Anwendungen, die von Kreativität profitieren (z.B. ein Chatbot), kann Sampling Vorteile bringen (Gante, 2023).

Mit der Version v4.37.0 (veröffentlicht am 22. Januar 2024) wurden mehrere Ansätze im Bereich Assisted Generation erweitert. Dazu gehört *ngram speculation*, womit die LLM Generierung 2x-4x beschleunigt werden kann, ohne dass ein Assistenzmodell notwendig ist. Diese Methode nutzt aus, dass es Überschneidungen bei dem Input und Output von LLMs gibt. Dies können z.B. Institutionsnamen aus mehreren Wörtern oder Codeabschnitte sein. Eine Überschneidung aus N-Grammen ermöglicht es, diese während der Generierung direkt aus dem Input zu kopieren, anstatt die Tokens einzeln autoregressiv zu generieren (Saxena, 2024). Weiter wurde auch Speculative Sampling nach Leviathan et al. (2023) implementiert, womit Assisted Generation erweitert wird. Speculative Sampling kann mit dem Flag do_sample=True aktiviert werden (amyeroberts, 2024).

2.4.1. Wahl von k

Mit k wird definiert, wie viele Kandidaten vom Draft Modell generiert werden. Leviathan et al. (2023) und C. Chen et al. (2023) verwenden hierfür einen statischen Wert für k.

Zum Zeitpunkt dieser Arbeit wird in Transformers (v4.40.0) (LysandreJik, 2024) ein statischer Wert k initialisiert, der sich über die Iterationen abhängig von der vergangenen Performance ändert. Wurden in der letzten Iteration alle Draft Kandidaten akzeptiert, wird k um 2 inkrementiert. Falls mindestens ein Draft Kandidat nicht akzeptiert wird, dekrementiert sich k um 1. Damit Assisted Generation funktioniert, wird k nie kleiner als 1. Der schematische Ablauf dieser Strategie ist in der Abbildung 2.2 ersichtlich.

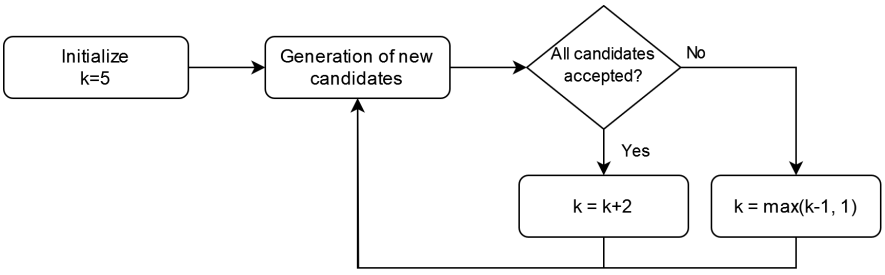


Abbildung 2.2.: Update-Schema von k in Transformers (v4.40.0)

Dieser Ansatz (+2/-1 Strategie) erlaubt es, die Anzahl an Kandidaten k der Schwierigkeit der Anwendung anzupassen. Als Beispiel: Es werden 4 Iterationen benötigt, um von k=5

auf k=1 zu kommen. Da das Update nur jede Iteration mit kleinen Änderungen erfolgt, ist die Strategie zu träge, um sich lokalen Schwankungen der Schwierigkeit anzupassen. Zudem ist k ausschliesslich abhängig von der vergangenen Performance. Auf Fehler wird somit erst reagiert, wenn sie bereits eingetreten sind. Erst beim Forward-Pass durch das Target Modell wird die Performance evaluiert.

3. Ideen und Konzepte

3.1. Adaptive Draft-Länge basierend auf Entropie

In beiden Papers, Leviathan et al. (2023) sowie C. Chen et al. (2023) wird ein Draft Modell verwendet, um die Kandidaten zu generieren. Das Draft-Modell wird zur Bildung eines Kandidaten k-mal aufgerufen, bevor der ganze Kandidat über das Target-Modell validiert wird. Die Wahl eines geeigneten Werts für k (bzw. γ in Leviathan et al. (2023)) wird in den Papers mathematisch oder durch Experimente eruiert. Zwecks Einheitlichkeit wird nachfolgend der Begriff k verwendet.

In C. Chen et al. (2023) wird ein optimaler Faktor für k eruiert und für alle nachfolgenden Experimente verwendet. In Leviathan et al. (2023) werden verschiedene Werte in den Experimenten verwendet. In beiden Fällen wird k vor dem Experiment definiert und ändert sich während der Generierung nicht.

Es wird angenommen, dass die Schwierigkeit von Token Predictions über den Verlauf eines generierten Texts variiert. Demnach sind nicht alle Abschnitte gleich einfach vorherzusagen. Als Beispiel: Einfache Abschnitte könnten oft verwendete Redewendungen oder bei einem Codegenerator Boilerplate-Code sein. Schwierigere Abschnitte erfordern tieferes Verständnis der Zusammenhänge und des Kontexts. Mit einem statischen k wird ein Kompromiss zwischen einfachen und schwierigen Textstellen eingegangen.

Mit einem höheren Wert für k wird das Target-Modell weniger aufgerufen. Dies führt zu höherer Effizienz und beschleunigter Generierung. Allerdings sinkt mit höherem k der Anteil an akzeptierten Tokens. Je länger der Kandidat, desto wahrscheinlicher, dass ein Token abgelehnt wird und dieser mit allen nachfolgenden Tokens verworfen wird. Dies führt zu einer Trade-Off Situation. Einerseits ist es erstrebenswert, möglichst viele Kandidaten zu generieren, andererseits sollte die Akzeptanzrate so hoch wie möglich gehalten werden.

In Leviathan et al. (2023) wird weitergehende Forschung an adaptiven Strategien für die Wahl von k motiviert. „There are several directions for follow up research [...] In this work we fixed the approximation model and the number of guesses γ throughout inference, but varying them during inference could yield additional improvements.“ (Leviathan et al., 2023, S. 8)

Dieser Ansatz stützt sich darauf, schon während der Generierung die Unsicherheit des Modells messen und auszuwerten. Anstatt in jeder Iteration eine fixe Anzahl an Tokens zu generieren, kann die Drafting Phase beendet werden, sobald die Unsicherheit steigt. Dafür wird eine zuverlässige Metrik vorausgesetzt, da wir die Validierung mit dem Target Modell erst am Ende der Drafting Phase aufrufen. In der Informationstheorie quantifiziert Entropie den Informationsgehalt einer Variable. Je tiefer die Entropie, desto tiefer ist der Informationsgehalt (Vajapeyam, 2014). In diesem Fall ist der Ausgang berechenbarer. Für jeden Generierungsschritt gibt es eine Wahrscheinlichkeitsverteilung für Tokens. Aus dieser lässt sich die Entropie berechnen.

Damit ist es möglich, für jedes Token den Überraschungsgehalt der Vorhersage zu berechnen, ohne jegliche externe Evaluation. Allerdings ist unklar, ob mit dieser Metrik zuverlässig vorausgesagt werden kann, ob ein Token akzeptiert wird oder nicht. Anders gesagt: Werden Tokens mit niedriger Entropie eher akzeptiert?

Sollte diese Hypothese bestätigt werden, können Strategien entwickelt werden. Diese Strategien entscheiden während der Drafting Phase in jeder Iteration, ob das Drafting fortgeführt werden soll oder ob der Kandidat für das Scoring zurückgegeben wird. Die Strategien können anhand von Daten von Testläufen evaluiert werden.

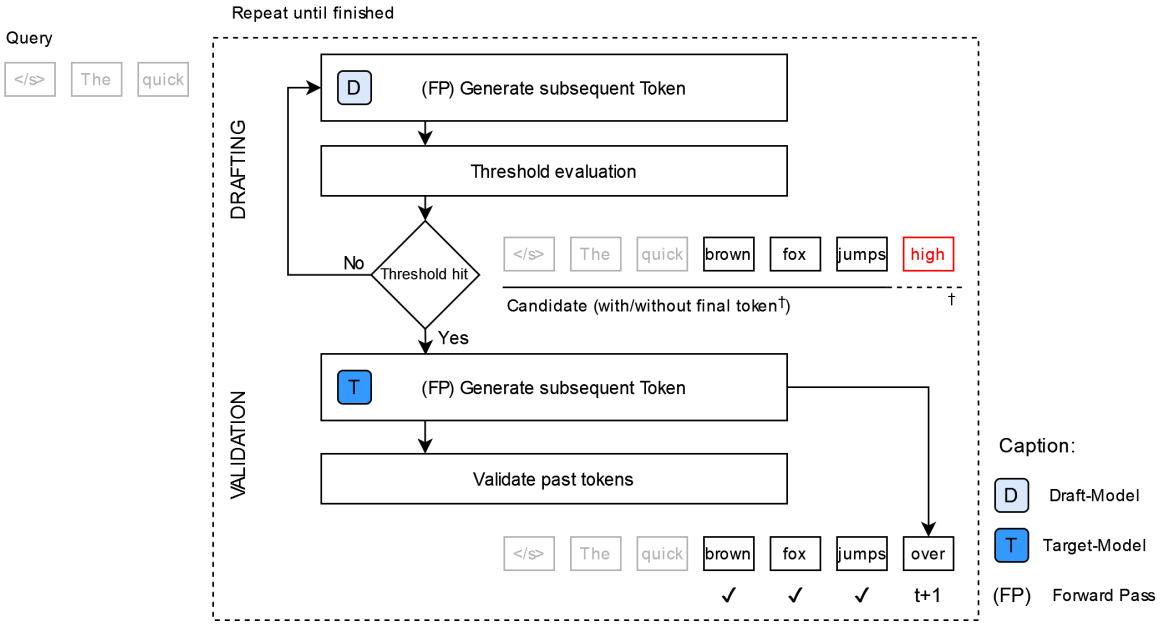


Abbildung 3.1.: Schematische Darstellung einer Iteration Text-Generierung bestehend aus den Phasen Drafting (1) und Validation (2) (Idealszenario)

In Abbildung 3.1 ist der Ansatz schematisch dargestellt. Die Generierung besteht aus den Phasen Drafting und Validation. Diese Idee betrifft nur den Drafting Prozess. Anstelle einer statischen Zahl k Kandidaten wird die Generierung so lange fortgesetzt, bis der Grenzwert der Strategie erreicht wird. Danach wird der Kandidat der Validation übergeben.

4. Methodik

4.1. Projektplanung

Das Projekt wurde in drei Hauptphasen aufgeteilt: Initialisierung, Durchführung und Kontrolle/Auswertung. Diese Aufteilung orientiert sich an dem vorgegebenen Zeitplan für die Bachelorarbeit. Die rot markierten Felder markieren Meilensteine. Meilensteine stehen für planungsrelevante Ereignisse z.B. Präsentationen, Abgabe von Lieferobjekten.

Der Fokus in der Initialisierungsphase liegt darin, ein Grundverständnis über die Aufgabenstellung und die dafür relevante Literatur zu verschaffen und wird mit der Signatur der Aufgabenstellung abgeschlossen. Die Durchführungsphase ist der Hauptteil dieser Arbeit und besteht aus Vorbereitung, Durchführung und Auswertung aller Experimente sowie dem Verfassen sämtlicher Projektartefakte. In der abschliessenden Phase Kontrollieren / Auswerten wird das komplette Projekt inklusive der erzielten Ergebnisse reflektiert.

Die Projektplanung 4.1 ist ein Kontrollwerkzeug zur laufenden Abschätzung und Einteilung des Aufwandes. Die Planung hilft dabei, beim wöchentlichen Review den aktuellen Fortschritt zu überwachen und eine Priorisierung vorzunehmen. Die agile Vorgehensweise erlaubt es, über die ganze Projektdauer Änderungen an der Planung vorzunehmen. Dazu gehört zum Beispiel das Erfassen von neuen Tasks.

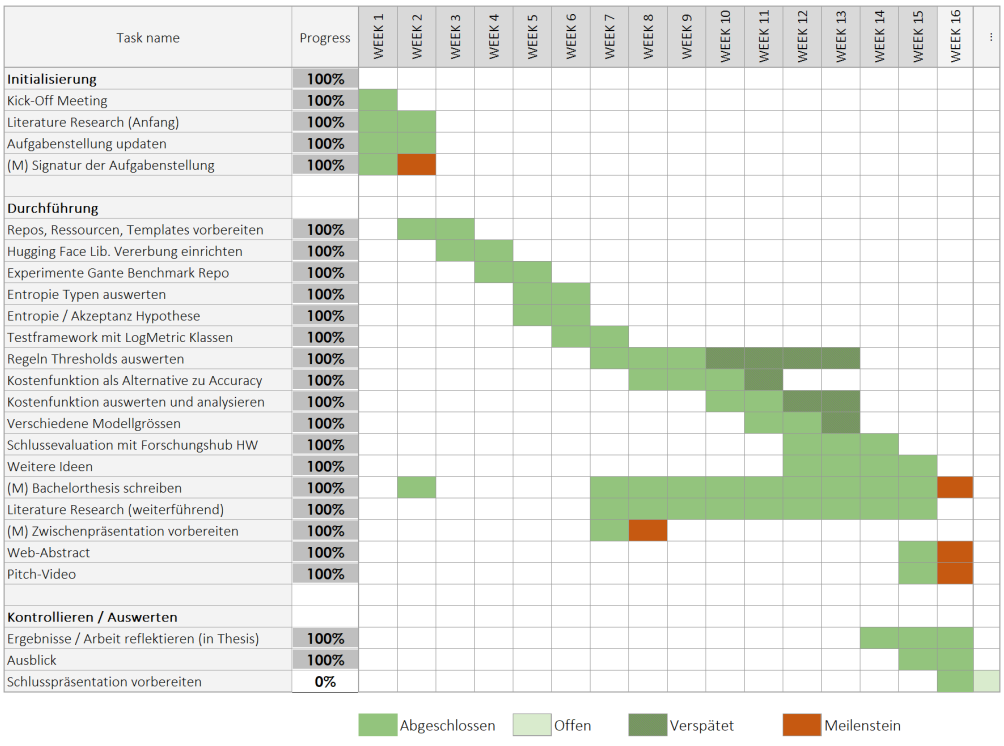


Abbildung 4.1.: Gantt Projektplanung

4.2. Projektmanagement

In diesem Projekt wurde eine agile Vorgehensweise angewandt. Auf ein Wasserfallmodell mit voneinander abgegrenzten Projektphasen wurde aus den folgenden Gründen verzichtet.

- Planungsunsicherheit: Aufgrund des Forschungscharakters der Arbeit ist eine Detailplanung zu Beginn nicht gut möglich. Das weitere Vorgehen ist abhängig von den erzielten Ergebnissen und nicht von Beginn absehbar.
- Flexibilität: Durch die wöchentlichen Reviews können Prioritäten laufend und basierend auf dem aktuellen Fortschritt und den Ergebnissen gesetzt werden
- Feedback: Hohe Gewichtung des Feedbacks durch die Reviews

Es findet jede Woche ein Review mit der Betreuungsperson statt. In dieser werden angefallene Erkenntnisse und Ergebnisse besprochen und nächste Schritte besprochen. Zudem wird der aktuelle Fortschritt reflektiert.

Im Gegensatz zu agilen Softwareprojekten wird kein iterativer Release/Build bereitgestellt. Zudem wird der Projektmanagement-Overhead aufgrund des Projektumfangs bemessen gehalten.

4.3. Forschungsmethodik

Zu Beginn wird eine Literaturrecherche durchgeführt, um die aktuellen Methoden mit ihren Vor- und Nachteilen und Herausforderungen besser zu verstehen. Die Literaturrecherche fokussiert sich nicht auf eine spezifische Forschungsfrage (wie bei einer systematischen Literaturrecherche). Es wurde eine Literaturrecherche des Typs Scoping Review angewendet, um eine Momentaufnahme des Forschungsfeldes zu machen. Scoping Reviews konzentrieren sich auf Schlüsselkonzepte eines Forschungsbereichs. Es werden die verfügbaren Methoden, Evidenz und insbesondere auch Forschungslücken festgestellt (Xiao & Watson, 2019). Für das Verfolgen von einem entropiebasierten Ansatz, muss abgeklärt werden, ob die Entropie ein zuverlässiger Prädiktor für die Akzeptanz eines Tokens ist. Diese Hypothese wird durch ein Experiment mittels statistischer Analyse überprüft. Wird die Hypothese bestätigt, kann der Ansatz weiter verfolgt werden. Es wird ein entropiebasierter Ansatz implementiert. Durch die Evaluation wird festgestellt, ob der Ansatz zu einer signifikanten Beschleunigung der Generierungszeit führt. Zudem wird festgestellt, welche Hyperparameter bei den Strategien zu den besten Ergebnissen führen. Es werden Experimente in verschiedenen Settings durchgeführt, um die Performance zu messen.

Die vorliegende Arbeit basiert ausschliesslich auf öffentlich zugänglichen Quellen. Dies umfasst wissenschaftliche Publikationen, Programmcode, Datasets sowie Benchmarks. Alle Quellen sind über das Literaturverzeichnis referenziert.

Alle Implementationen werden auf Basis der Transformers-Bibliothek von Hugging Face (Wolf et al., 2020) vorgenommen. Die Codebase wird auf einem nicht-öffentlichen Git Repository auf gitlab.switch.ch gehalten. Als Entwicklungsumgebung wird VS-Code verwendet. Die vorliegende Arbeit wird mit LaTeXverfasst.

4.4. Nutzung von AI Tools

Für diese Arbeit wurden folgende AI-Tools als Unterstützung verwendet.

- ChatGPT: Übersetzungen, Synonyme, Paraphrasierungen, Zusammenfassung von Papers
- GitHub Copilot: Code-Assistent
- Elicit: Suche für relevante Papers

5. Realisierung

5.1. Modifikation der Transformers Library

Als Codebasis für diese Arbeit wird die Library Transformers von Hugging Face verwendet. Transformers hat sich im NLP-Bereich etabliert und ist eine umfassende Toolsammlung, um vortrainierte Modelle für verschiedenste Anwendungen einzusetzen. Die Library unterstützt Jax, PyTorch und TensorFlow als Deep Learning Libraries und ist eng verknüpft mit dem Hugging Face Hub. Der Hub ist eine Plattform, um Modelle, Datasets und Weiteres zu teilen (Wolf et al., 2020).

Für diese Arbeit von Interesse sind vor allem die Generation Tools. Dazu gehören sämtliche Klassen und Methoden, die an der Generierung beteiligt sind. Durch Debugging können relevante Klassen gefunden werden.

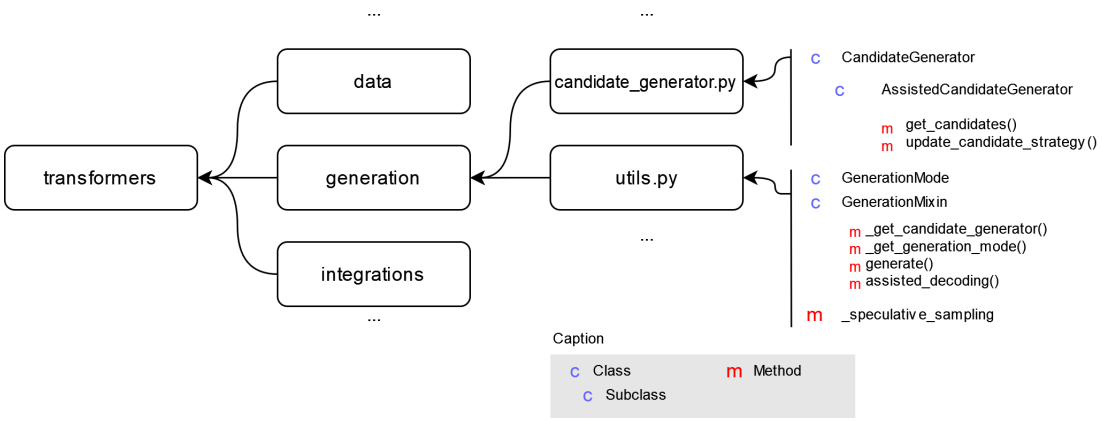


Abbildung 5.1.: Grobübersicht über den Aufbau der Transformers Library mit den für diese Arbeit besonders relevanten Klassen und Methoden. (nicht abschliessend)

Die Änderung von Source Code in der Library ist für dieses Projekt aus zwei Gründen notwendig.

Debugging und Logging:
Debugging über die Entwicklungsumgebung gewährt zur Laufzeit einen Einblick in den Programmablauf, ohne Änderungen an der Source. Die Auswertungsmöglichkeiten im Live Debugging sind allerdings limitiert und für eine grössere Zahl an Beobachtungen ungeeignet. Für diesen Zweck habe ich Logging Klassen mitgegeben, die beliebige Messungen machen (z.B. Entropie) und gebündelt als Objekt zurückgeben. Die Logging Klassen haben keinen Einfluss auf den Funktionsablauf.

Implementation von Projektideen:
Hier wird der Quellcode um die in dieser Arbeit beschriebenen Ideen ergänzt. Diese Änderungen haben Einfluss auf den Funktionsablauf und werden für Benchmarks und Auswertungen vorausgesetzt.

5.1.1. Subclassing

Die Änderungen von Source Code werden über Subclassing realisiert. Dies erlaubt es, das Verhalten von einzelnen Komponenten der Library zu überschreiben, ohne Änderungen direkt an der Source vorzunehmen.

Gemäss Illustration 5.2 kann eine Subklasse erstellt werden, die das Verhalten der Superklasse für einzelne Methoden überschreibt.

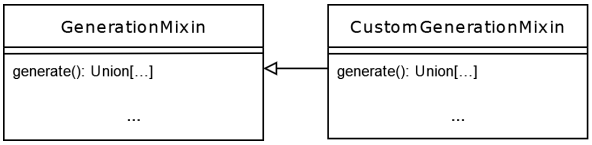


Abbildung 5.2.: Subclassing am Beispiel der Klasse GenerationMixin in UML Notation

Nach mehreren Versuchen war evident, dass der Ansatz mit Vererbung nicht zum erwarteten Resultat führte. Hierfür wurde eine Subklasse für *AutoModelForCausalLM* erstellt und mit Test-Methoden ergänzt. Diese Klasse lädt das Modell und implementiert den generate() Aufruf. Nach der Instanziierung waren die Test-Methoden nicht vorhanden.

Die gescheiterte Vererbung ist dadurch zu erklären, dass die Klasse AutoModelForCausalLM zum Zeitpunkt der Vererbung leer ist. Die Instanziierung, in diesem Fall über die Methode .from_pretrained(), löst Factory-Methoden aus, die das Objekt erst über diesen Aufruf bauen. Dies hat den Vorteil, dass über Mapping Scripts die passende Funktionalität gemäss Modelltyp zuweisen können.

5.1.2. Mixin Beimischung zu einer Instanz

In der objektorientierten Programmierung versteht man unter Mix-In ein Sprachkonzept, mit dem Funktionalität in andere Klassen gemischt werden kann. Im Gegensatz zu einer einfachen Vererbung kann die Mixin-Klasse während Laufzeit in beliebige Klassen gemischt werden (Esterbrook, 2001). Da Python Mehrfachvererbung unterstützt, kann Mixin hier verwendet werden.

Um die Klasse AutoModelForCausalLM mit einer eigenen Implementation für .generate() zu ergänzen, wird diese nach der Instanziierung als Objekt um eine Basisklasse ergänzt. Hierbei wird lediglich die Instanz des Objekts angepasst.

```
def extend_instance(obj, cls):
    """Apply mixins to a class instance after creation"""
    base_cls = obj.__class__
    base_cls_name = obj.__class__.__name__
    obj.__class__ = type(base_cls_name, (base_cls, cls),{})
```

Code-Snippet zur Mixin Beimischung nach Instanziierung (SleepyCal, 2015)

Über diesen Weg ist es möglich, eine eigene .generate() Methode zu injizieren und jeglichen folgenden Programmfluss in Subklassen zu leiten.

Es bleibt anzumerken, dass das Überschreiben der Basisklasse eines bereits instanziierten Objekts mit den Best Practices der objektorientierten Programmierung divergiert und nur mit Vorsicht angewendet werden sollte.

5.2. Effektivität von Entropie als Abbruchkriterium

In diesem Kapitel wird eruiert, ob die Entropie als Abbruchkriterium geeignet ist. Sind die Ergebnisse positiv, wird der Ansatz weiter verfolgt. Im Folgenden wird mit dem Begriff Entropie die Shannon-Entropie aus der Informationstheorie gemeint.

Die Shannon Entropie gibt in der Informationstheorie den Informationsgehalt einer Variable an. Damit beschreibt sie, wie viel Speicher eine Information (nicht die Daten davon) beansprucht. Informell ist der Informationsgehalt proportional mit dem Überraschungsgehalt einer Information. Je einfacher das Erraten einer Information ist, desto kleiner ist die Überraschung und somit auch die Entropie (Vajapeyam, 2014).

$$H(I) = -\sum_{i \in I} p(i) * log(p(i))$$

Die Entropie einer Verteilung I kann über obenstehende Formel bestimmt werden. Als Beispiel für diese Anwendung werden zwei Token-Wahrscheinlichkeitsverteilungen mit je 3 Tokens angenommen. In I_1 hat das erste Token eine Wahrscheinlichkeit von 96 %, die anderen beiden haben je 2 %. In I_2 hingegen hat jedes der drei Tokens dieselbe Wahrscheinlichkeit. Bei I_1 ist sich das Modell viel sicherer als bei I_2. Es fällt auch wesentlich leichter, I_1 zu erraten. Dies widerspiegelt sich auch deutlich in der Entropie. I_1 hat eine Entropie von 0.282, während I_2 bei 1.585 liegt.

$$H(I_1) = -(0.96 * log_2(0.96) + 0.02 * log_2(0.02) + 0.02 * log_2(0.02)) \approx 0.282$$
$$H(I_2) = -(\frac{1}{3} * log_2(\frac{1}{3}) * 3) \approx 1.585$$

Für die Realisierung der Idee einer adaptiven Draft-Länge basierend auf Entropie 3.1 wird vorausgesetzt, dass ein Zusammenhang zwischen der Entropie und der tatsächlichen Akzeptanzwahrscheinlichkeit eines Tokens besteht. Falls die Entropie keine hilfreiche Information zur Akzeptanzwahrscheinlichkeit erbringt, ist es nicht erstrebenswert, diesen Ansatz weiter zu untersuchen.

Daraus ergibt sich die Hypothese: Tokens mit hoher Entropie werden eher abgelehnt

In Cai et al. (2024) wird *Typical Acceptance* präsentiert, ein Ansatz, der die Entropie der Verteilung in die Sampling-Strategie einbezieht. Dies wird damit begründet, dass neben der Token-Wahrscheinlichkeit auch die Entropie eine entscheidende Rolle spielt. Bei hoher Entropie können mehrere Tokens plausibel sein. Die Tatsache, dass Entropie für Sampling verwendet werden kann, legt nahe, dass sie nützliche Informationen beisteuert.

Die Hypothese wird empirisch über ein Experiment untersucht. Hierfür wird für eine Anzahl von $s = 1000$ Samples aus dem Datensatz c4 (en; validation) (Raffel et al., 2020) Text generiert. Als Draft- und Target-Modell (m_d / m_t) werden OPT-125m beziehungsweise OPT-2.7b verwendet. Es werden die Entropien berechnet. Zudem wird für beide Verteilungen (Draft und Target) die argmax Funktion ausgeführt. Dies wählt das Token mit der höchsten Wahrscheinlichkeit (Greedy Decoding) (Shi et al., 2024). Daraus ist erkennbar, ob das Draft-Modell

dasselbe Token wie das Target-Modell gewählt hat – oder davon abgewichen ist.

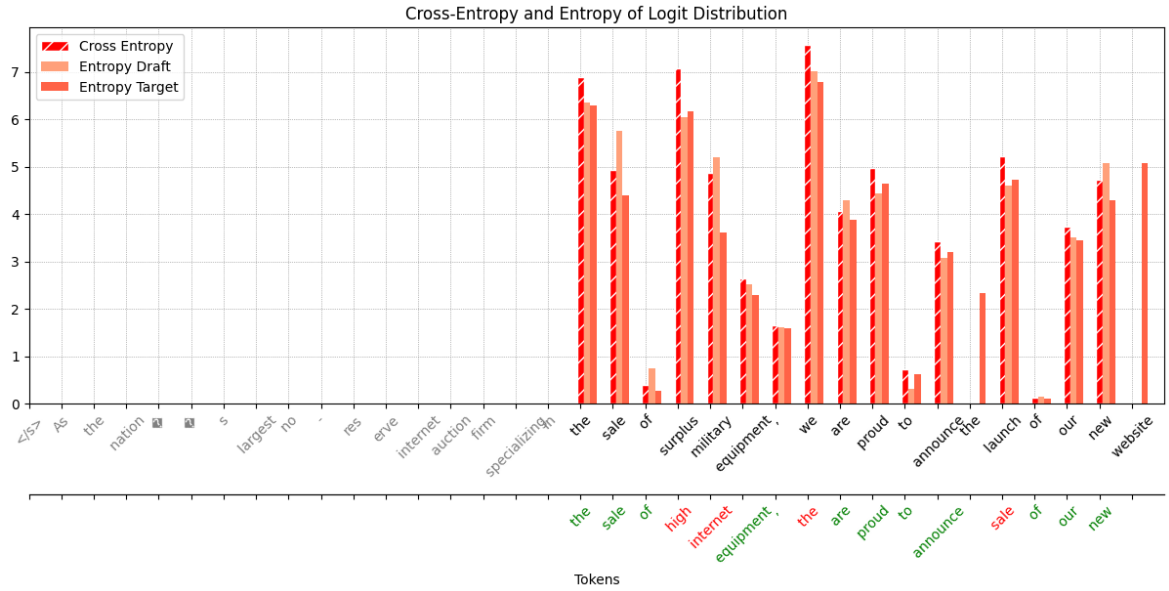


Abbildung 5.3.: Visualisierung der Entropien über eine Sequenz

In Abbildung 5.3 sind Entropien von Draft- und Target-Verteilung sowie die Cross-Entropie über beide Verteilungen sichtbar. Für diese Idee ist nur die Draft-Entropie relevant, da nur diese während der Drafting-Phase berechnet werden kann. Die grauen Tokens links sind Teil der Abfrage. Die Labels in der oberen x-Achse stellen die decodierten Tokens des Target-Modells dar. In der unteren x-Achse sind die decodierten Tokens des Draft-Modells. Die Farbcodierung zeigt an, ob der Draft-Vorschlag mit dem Target Token übereinstimmt.

Für eine aussagekräftige Analyse wird eine grössere Zahl an Samples ($s = 1000$) generiert und die Tokens mit ihrer Draft-Entropie in die Kategorien Matching und Non-Matching unterteilt. Das Verhältnis von Matching zu Non-Matching ist ungefähr 2:1.

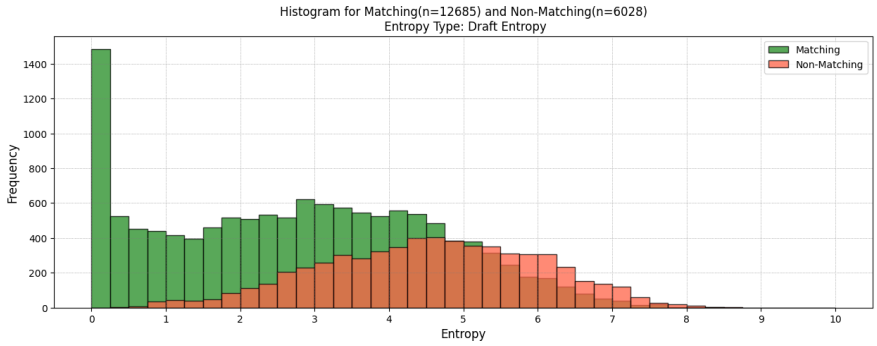


Abbildung 5.4.: Histogramm für n=18’713 Tokens aus s=1000 Samples

Das Histogramm 5.4 zeigt einen deutlichen Zusammenhang zwischen der Entropie (x-Achse) und der Akzeptanzhäufigkeit. Eine niedrige Entropie ist ein sehr starker Indikator, dass ein Token akzeptiert wird. Mit steigender Entropie nimmt die Akzeptanzrate ab. Die Resultate dieses Experiments stützen die Hypothese deutlich. Tokens mit einer Entropie unter 2 werden fast immer akzeptiert. Tokens mit einer Entropie über 5 werden zunehmend abgelehnt.

5.3. Threshold Strategien

In Kapitel 5.2 wurde bekräftigt, dass die Entropie eines Tokens auf dessen Akzeptanzwahrscheinlichkeit schliessen lässt. Diese Annahme ist Grundlage für die Entwicklung von Threshold Strategien. Threshold Strategien entscheiden nach jedem generierten Draft-Token, ob die Drafting Phase abgeschlossen werden soll oder ob ein nächstes Token generiert werden soll.

Eine optimale Threshold Strategie generiert genau so viele Draft-Tokens in einer Drafting-Phase, wie vom Target-Modell akzeptiert werden. Somit kann das volle Potenzial von Assisted Generation entfaltet werden. Ist die Threshold Strategie zu strikt, wird die Drafting Phase beendet, selbst wenn das nächste Draft-Token noch akzeptiert worden wäre. Ist die Threshold Strategie hingegen zu locker, werden unstimmige Kandidaten geliefert, die vom Target Modell verworfen werden. Die Threshold Strategien werden auch Regeln genannt.

5.3.1. Static Threshold

Als einfachste der Threshold Strategie prüft der *Static Threshold* die Entropie x_t mit einem gegebenen Threshold τ. Wird der Threshold erreicht oder überschritten, greift die Regel. Der Static Threshold ignoriert vergangene Werte.

Parameter:

- τ: Threshold (Entropie)

$$\tau \leq x_t$$

5.3.2. Moving Average

Der *Moving Average* ist ein relativer Threshold. Anstelle der absoluten Entropiewerte ist der Trend ausschlaggebend. Es wird der Mittelwert für die letzten n quadrierten Entropien der laufenden Drafting Phase bestimmt. Multipliziert mit dem Faktor λ ergibt sich daraus der Threshold. Wenn das Quadrat der aktuellen Entropie x_t den Threshold erreicht oder überschreitet, greift die Regel.
Parameter:

- n_{max}: Maximal auszuwertende vergangene Entropien in laufender Drafting Phase
- λ: Threshold-Faktor

$$\frac{1}{n} \sum_{i=1}^{n_{max}} x_{t-i}^2 * \lambda \leq x_t^2$$

Im Vergleich zu einem gewöhnlichen Moving Average gibt es zwei Änderungen. Einerseits werden alle Werte quadriert. Dies ist inspiriert von *Sum of Squared Errors* (SSE), wobei grössere Fehler durch die Quadrierung hervorgehoben werden („Least-Squares Method“, 2008). Bei der zweiten Änderung handelt es sich um den Threshold-Faktor λ. Mit λ kann der aus den vergangenen Entropien resultierende Threshold justiert werden. Ein optimaler Wert kann durch Tuning eruiert werden.

Es werden nur Entropien aus der aktuellen Drafting-Phase berücksichtigt. Dies verhindert, dass grenzüberschreitende Entropien aus vergangenen Iterationen nochmals ausgewertet werden. Ohne diese Massnahme könnte eine hohe Entropie nach Stopp der eigenen auch zum Stopp der nächsten Drafting Phase führen. Wie in Abbildung 5.4 ersichtlich, werden Tokens

mit niedriger Entropie fast immer akzeptiert, unabhängig von den vergangenen Entropien. Die Limitation auf Tokens aus der aktuellen Drafting-Phase führt allerdings auch dazu, dass in vielen Fällen keine oder nur wenige vergangene Tokens verfügbar sind.

Zur Bestimmung vom Moving Average ist mindestens ein vergangenes Token notwendig. Damit kann die Regel erst für Positionen ≥ 2 berechnet werden. n_{max} bestimmt die maximale Anzahl vergangener Entropien, die in die Regel einfliessen.

5.3.3. Cumulative Threshold

Der *Cumulative Threshold* ist eine Erweiterung vom Static Threshold 5.3.1. Bei dieser Regel wird die Summe aus der aktuellen Entropie mit den n letzten quadrierten Entropien berechnet. Die Entropien werden aus denselben Gründen wie in Moving Average 5.3.2 beschrieben quadriert, um die Fehler hervorzuheben. Erreicht oder überschreitet diese Summe den gegebenen Threshold τ, greift die Regel.

Parameter

- n_{max}: Maximal auszuwertende vergangene Entropien in laufender Drafting Phase
- τ: Kumulierter Threshold (Entropie)

$$\tau \leq \sum_{i=0}^{n_{max}} x_{t-i}^2$$

Cumulative Threshold summiert bis zu n_{max} vergangenen Entropien und kann für alle Positionen ≥ 1 bestimmt werden.

5.4. Kostenfunktion

Der Zweck einer Kostenfunktion ist es, die Leistung eines Modells zu quantifizieren. Durch Änderung von Parametern kann das Modell optimiert werden. Die Kostenfunktion ist im Rahmen dieser Arbeit notwendig, um die optimalen Hyperparameter zu finden und die Performance evaluieren.

Ein möglicher Ansatz ist es, die Genauigkeit zu messen. Durch das Aufstellen einer *Confusion Matrix* können sämtliche Genauigkeitsmetriken wie Accuracy, Precision, Recall oder F1-Score berechnet werden. Als Beispiel kann die Regel gefunden werden, die Drafting-Phasen am verlässlichsten abschliesst. Ein gravierender Nachteil dieser Methode ist, dass sie von den im Experiment verwendeten Einstellungen (v.a. Modellgrössen) abhängig ist. Zudem ist eine Genauigkeitsmessung für diese Anwendung nicht zweckmässig, da Assisted Generation verlustfrei generiert – lediglich unter verschiedenem Aufwand.

Naheliegender ist eine ressourcenorientierte Kostenfunktion. Ressourcen sind einerseits Rechenkapazität, andererseits Zeit. Möglich wäre eine Messung der *Floating point operations per second* (FLOPS) des Grafikprozessors. Wie in der Problemstellung 2.1 erläutert, sind arithmetische Operationen in der Regel nicht die Limitation bei autoregressiver Generierung. Daher wird dieser Ansatz nicht weiter verfolgt. Am vielversprechendsten ist es, Zeit als Kostenfunktion zu verwenden. Somit können die Ergebnisse mit ähnlichen Arbeiten (Leviathan et al. (2023), Gante (2023)) verglichen werden, welche sich auf Beschleunigung und Latenz der Inferenz konzentrieren.

$$c = x_d * t_d + x_t * t_t$$

Die Kostenfunktion ergibt sich aus der Anzahl Aufrufen x und der benötigten Zeit für einen Forward-Pass t. c entspricht der Zeit in Millisekunden, die für eine Generierung einer Sequenz notwendig ist. Diese Definition bedeutet, dass c stark abhängig von der verwendeten Hardware, den verwendeten Modellen oder Einstellungen der Generierung (z.B. Sequenzlänge) ist. Experimente mit unterschiedlichen Einstellungen können nicht verglichen werden. Wie in Leviathan et al. (2023) wird für die Evaluation ein Faktor berechnet, da eine relative Metrik viel besser generalisierbar ist.

5.5. Hyperparameter Suche

In diesem Kapitel wird die Suche nach optimalen Hyperparametern für die Threshold Strategien 5.3 beschrieben. Dabei wird eruiert, welche Threshold Einstellungen die Kostenfunktion 5.4 am meisten minimieren. Das ermöglicht es, den Ansatz im weiteren Verlauf der Arbeit mit verschiedenen Methoden zu evaluieren und vergleichen.

Die Hyperparameter Suche verläuft in zwei Stufen. Erst wird die Inferenz auf Basis von aufgezeichneten Samples simuliert. Das Ziel dieser Simulation ist es, die Kosten unter gegebenen Regeln für ein Sample abzuschätzen. Auf diese Weise können die Kosten für eine grosse Bandbreite an Hyperparametern effizient abgeschätzt werden. Damit kann der Suchbereich stark eingegrenzt werden. Die interessanten Hyperparameter-Bereiche werden im zweiten Schritt gemessen, indem mit den implementierten Strategien generiert wird.

Je nach Auswahl der Ranges kann es über 100 zu testende Parameter-Einstellungen für eine Strategie geben. Unter den Samples gibt es teilweise beträchtliche Variabilität in den Punkten Schwierigkeit, Text-Typ und Thema. Daher ist es notwendig, jede Einstellung mit vielen Samples zu testen, um daraus die Durchschnittskosten zu berechnen.

Der ausschlaggebende Grund für eine Simulation sind die hohen Berechnungszeiten für Inferenz. Tabelle 5.1 vergleicht die Berechnungskosten von Simulation und Inferenz für 1000 Samples.

Methode	Zeit
Simulation	< 0.5s
Inferenz	≈ 500s

Tabelle 5.1.: Berechnungszeit Kosten einer Hyperparameter-Einstellung über 1000 Samples

Die Berechnungszeit für Inferenz in Tabelle 5.1 bezieht sich auf die in diesem Projekt verwendete Hardware und ist weiter von den Modellen, den Generation-Settings sowie von Regelparametern abhängig. Der Einfachheit halber wird von einer durchschnittlichen Berechnungszeit von 0.5s pro Sample ausgegangen.

Mithilfe der Simulation kann eine Approximation des Suchraumes in 1/1000 der Zeit abgebildet werden. Die tatsächlichen Kosten können erst im zweiten Schritt über die Inferenz bestimmt werden.

5.5.1. Versuchsaufbau

Dieses Kapitel beschreibt den Versuchsaufbau und verwendeten Einstellungen, die für die nachfolgenden Experimente verwendet wurden. Die verwendeten Einstellungen werden in Tabelle 5.2 gelistet.

Kategorie	Einstellung	Wert
Hardware und Modelle	GPU	NVIDIA RTX A4000 (16GB)
	Target Modell	OPT-2.7B
	Draft Modell	OPT-125M
	Präzision	FP32
Generierungsparameter	do_sample	False
	max_new_tokens	25
	no_repeat_ngram_size	6
Datensatz und Prompt	Task	Text Generation
	Dataset	c4;en;validation
	Loading Method	Streaming
	Input Tokens	25
Decoding	Decoding	Greedy (argmax)

Tabelle 5.2.: Systemparameter für Hyperparameter Suche

Das Modell OPT ist in verschiedenen Grössen von 125 Millionen bis zu 175 Milliarden Parametern verfügbar (Zhang et al., 2022). Die Effektivität von Assisted Generation ist bei der Verwendung von möglichst unterschiedlichen Modellgrössen am grössten (Gante, 2023). Dies motiviert den Gebrauch eines möglichst grossen und teuren Target-Modells zusammen mit einem möglichst kleinen und günstigen Draft-Modells.

Ohne den Einsatz von Quantisierung ist OPT-2.7B die grösstmögliche Variante, die zusammen mit dem Draft-Modell (OPT-125M) geladen werden kann. Die Limitation ist der verfügbare Speicher der Grafikkarte.

Als Datensatz für die Generation Inputs wird der Colossal Clean Crawled Corpus (C4) Datensatz verwendet. Der Datensatz ist eine Sammlung an extrahierten Texten aus dem Web und wurde umfassend gefiltert, um unerwünschte Inhalte (z.B. Source Code, Cookie-Mitteilungen) zu entfernen (Raffel et al., 2020). Für diese Arbeit wurde das englische Subset von C4 verwendet.

Über no_repeat_ngram_size kann verhindert werden, dass sich N-Gramme ab einer bestimmten Grösse wiederholen („configuration_utils.py at v4.40.2“, 2024). Die Einstellung verhindert insbesondere, dass die Generierung ganze Sätze endlos wiederholt. Der Wert von sechs wurde gewählt, dass kleinere N-Gramme (z.B. Organisationsnamen aus mehreren Wörtern) wiederholt werden dürfen.

Eruierung von t_d und t_t

Die Kostenfunktion 5.4 ist abhängig von der benötigten Zeit für einen Forward-Pass im Draft bzw. Target Modell. Somit haben die verwendete Hardware sowie die gewählten Modelle erheblichen Einfluss auf die Kostenfunktion. Die Forward-Pass-Zeit variiert auch unter Iterationen. Der Einfachheit halber werden t_d und t_t für den weiteren Verlauf fixiert. Die Modellgrössen entsprechen dem Versuchsaufbau. 5.5.1

Dafür wird eine Aufzeichnung von 1000 generierten Samples inklusive der gemessenen Zeiten für Draft- und Target Generierungen gemacht. Die Durchschnittszeiten finden sich in Tabelle 5.3.

Modell	Mean	Standardabweichung
Draft	6.54 ms	0.56 ms
Target	34.15 ms	1.99 ms

Tabelle 5.3.: Berechnungszeiten (RTX A4000, m_d = OPT-125M, m_t = OPT-2.7B)

Die Bestimmung von Durchschnittswerten für t_d und t_t ermöglicht es, die Kosten für die Generierung abzuschätzen. Der Einfachheit halber werden die Werte gerundet.
($t_d = 7$ms, $t_t = 34$ms) Auf Grundlage dieser Daten können Simulationen erstellt werden.

5.5.2. Simulationsbasierter Suchgang

Die Simulation ermöglicht eine effiziente Abschätzung der Kosten für eine gegebene Hyperparameter-Einstellung. Als Basis davon wird die Inferenz für s Samples mit einer statischen Anzahl an Kandidaten (num_k) aufgezeichnet. Die Wahl von einem statischen num_k hat gegenüber einer produktiven Strategie (e.g. Hugging Face +2/-1 Strategie) den Vorteil, dass Entropie-Lücken reduziert werden können, indem eine Mindestlänge für Drafts festgelegt wird. Entropie-Lücken entstehen, wenn alle Kandidaten akzeptiert wurden und das folgende Token nur vom Target Modell berechnet wird. Für dieses Token fehlt dann die Draft-Entropie.

Die Entropie-Lücken werden durch Imputation mit der Target-Entropie ersetzt. Wie im Entropieverlauf 5.3 aufgezeigt, sind die Entropien von Draft und Target ähnlich. Daher eignet sich die Target-Entropie als Substitut.

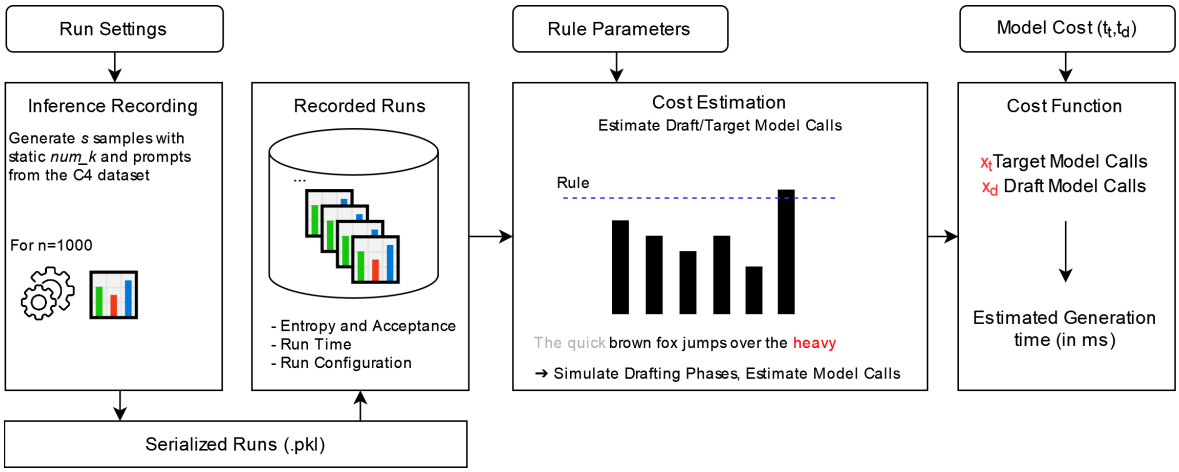


Abbildung 5.5.: Workflow Simulation

Der Workflow für die Simulation 5.5 besteht aus 3 Schritten: (1) Die Aufzeichnung von Samples, (2) Schätzung der Aufrufe und (3) Berechnung der Kosten. Der modulare Aufbau bezweckt Wiederverwendbarkeit und vermeidet Code-Redundanz. Durch die Aufzeichnungen wird vermieden, dass für jede Auswertung neu generiert werden muss.

Limitationen

Die grundlegende Schwierigkeit dieser Abschätzung ist folgende: Bei einer tatsächlichen Inferenz verändern sich die Drafting-Phasen, was für jede Regel zu einer neuen Entropie Ver-

teilung führt. In dieser Simulation wird stets von derselben Entropie ausgegangen. Die angewendeten Regeln beeinflussen die zukünftige Verteilung nicht.

Eine weitere Schwierigkeit ist das Abschätzen von grösseren Modellen. Wie im Workflow 5.5 gezeigt, können Modellkosten t_d und t_t für beliebige Modellgrössen in die Kostenfunktion gegeben werden. Allerdings kann aufgrund von Hardware-Limitationen 5.2 nicht mit Modellen generiert werden, die den verfügbaren Speicher übersteigen. Daher werden bei der Simulation von grossen Modellen lediglich die Kosten berücksichtigt und nicht allfällige Auswirkungen auf die Generierungsqualität. Das Abschätzen von grösseren Modellen ist interessant, weil Assisted Generation den besten Performancevorteil bei möglichst grossen Modellunterschieden gezeigt hat (Gante, 2023).

5.5.3. Inferenzbasierter Suchgang

Für den inferenzbasierten Suchgang werden die Threshold Strategien 5.3 in die Hugging Face Library implementiert. Hierfür wird der *CandidateGenerator* erweitert. Im Folgenden wird diese Implementation als Entropy Drafting bezeichnet. Der Name ergibt sich aus der Kernidee, eine Drafting Phase früher zu stoppen, falls das Modell unsicher wird.

```
entropy_drafting_params = {
    "type": "cum",
    "static_threshold_val" : 5,
    "ma_m" : 1.2,
    "ma_last_n" : 3,
    "cum_threshold_val" : 50,
    "cum_last_n" : 2,
}

tokens, dml, tml = model.generate(
                   **input_tokens,
                   assistant_model=assistant_model,
                   do_sample=False,
                   max_new_tokens=25,
                   no_repeat_ngram_size=6,
                   num_k=100,
                   entropy_drafting=True,
                   entropy_drafting_params=entropy_drafting_params,
                   )
```

Listing 5.1: Exemplarischer .generate() Aufruf

Der *.generate()* Aufruf wurde um drei Parameter erweitert, damit der Ansatz aktiviert und parametrisiert werden kann. Wird num_k spezifiziert, erfolgt die Generierung mit einer statischen Anzahl an Kandidaten. Die Hugging Face +2/-1 Strategie wird somit überschrieben. Mit entropy_drafting und entropy_drafting_params wird der Ansatz mit den Threshold Strategien erweitert. Mit entropy_drafting_params kann ein Dictionary mit den Hyperparametern mitgegeben werden.

Die Klasse wurde erweitert, dass als Rückgabewert neben Tokens auch ein *DecodingMetricsLogger* Objekt und ein *TimeMetricsLogger* Objekt zurückgegeben wird. In Ersterem befinden sich Daten zur Entropie und der Akzeptanz, in Letzterem Berechnungszeiten für das Target- und Draft-Modell.

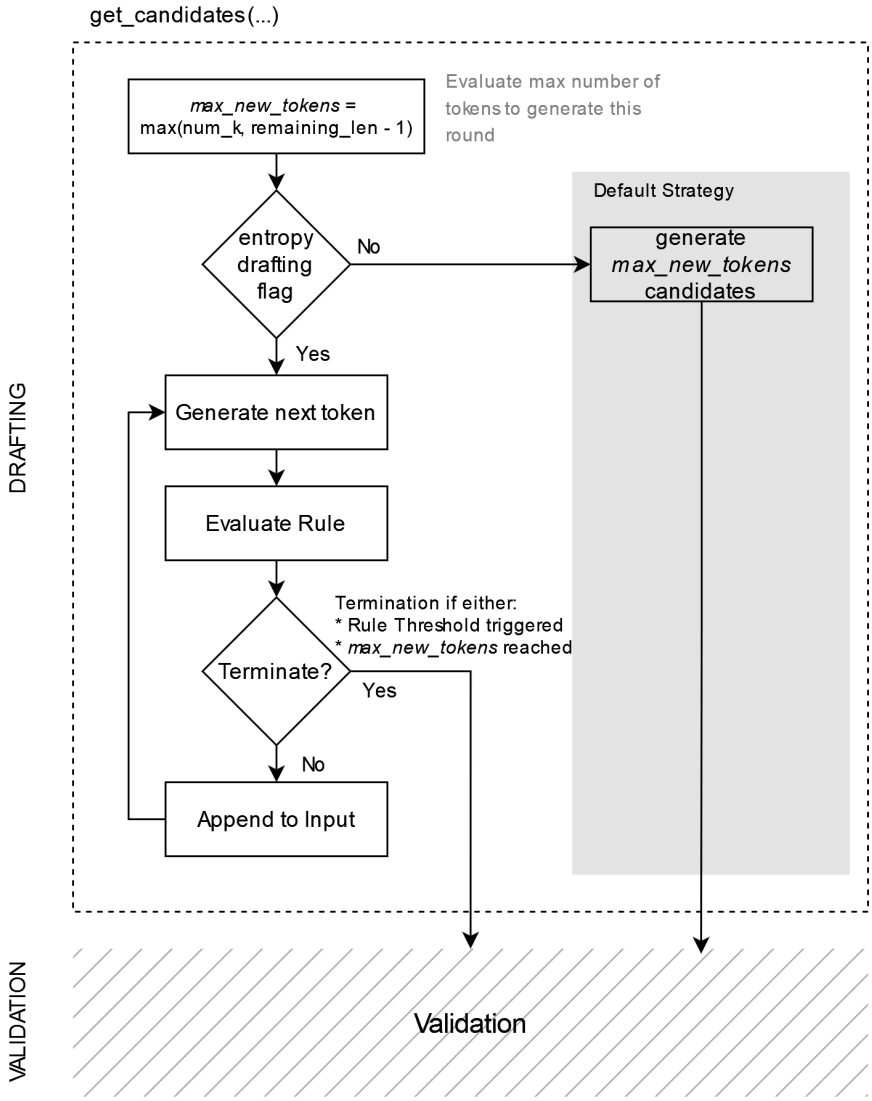


Abbildung 5.6.: Schematischer Ablauf von Entropy Drafting im CandidateGenerator

In Abbildung 5.6 wird der Ablauf der Implementation dargestellt. Die linke Seite des Diagramms zeigt die iterative Generierung von Entropy Drafting bis zum Eintritt eines Abbruchkriteriums. Die Default-Strategie (rechts) wurde beibehalten, damit Baseline-Messungen möglich sind.

Kostenberechnung in Inferenz

Ein wesentlicher Vorteil der Inferenz gegenüber der Simulation ist, dass die Kosten direkt aus der Laufzeit gemessen werden können. Dies ermöglicht eine Auswertung anhand der effektiv angefallenen Kosten in Laufzeit. Als Konsequenz werden auch Mehrkosten durch die Implementation mitgezählt.

Die Kosten können auch ähnlich wie bei der Simulation mit der Kostenfunktion anhand der Modell-Aufrufe mit t_d und t_t berechnet werden. Durch die Fixierung der Modell-Kosten werden die Mehrkosten der Implementation ignoriert. Dies könnte bei einer teuren Implementation nützlich sein, um festzustellen, ob der Ansatz sich bei einer Optimierung der Implementation lohnt.

Es ergeben sich also zwei Kosten für die Inferenz: Die Messung und die Kostenfunktion mit fixierten t_d und t_t. Für dieses Projekt wird ausschliesslich die Messung ausgewertet. Der Overhead der Implementation wurde gemessen und ist nicht hoch genug, dass Abschätzungen notwendig wären.

Finales Token im Kandidaten verwerfen

Bei der Rückgabe des Kandidaten in die Validation stellt sich die Frage, ob das letzte Token auch mitgegeben oder verworfen werden soll. Wie in Abbildung 5.6 gezeigt, wird ein Kandidat abgeschlossen, wenn entweder die maximale Anzahl an neuen Tokens erreicht wird oder der Threshold überschritten wird. In der überwiegenden Mehrheit der Fälle wird ein Kandidat wegen Überschreitung des Thresholds abgeschlossen. Daher stellt sich die Frage, ob das letzte und mutmasslich falsche Token verworfen werden soll.

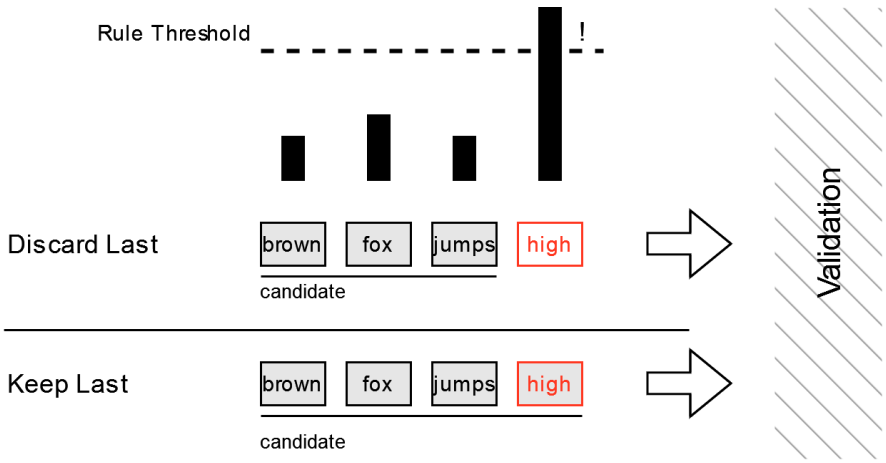


Abbildung 5.7.: Kandidat mit/ohne des finalen Tokens

Ein komplett korrekter Kandidat ist erstrebenswert. Werden bei der Validation alle Tokens im Kandidaten akzeptiert, kann der nachfolgende Token vom Target Modell verwendet werden. Dieser entsteht in der gleichen Operation der Validation als Nebenprodukt und ist somit kostenlos. Es gibt allerdings auch Gründe, die für die Verwendung des finalen Tokens sprechen. Die Erkennung über den Threshold ist nicht perfekt. Ein Teil der verworfenen Tokens wäre akzeptiert worden. Diese False Positives verursachen unnötige Kosten.

Ob die Anwendung dieser Regel sinnvoll ist, kann aufgrund vieler einfliessender Faktoren (Draft-Längen, Qualität der Drafts, ...) schwer abgeschätzt werden. Aus diesem Grund wird

ein Experiment in Inferenz gemäss Versuchsaufbau 5.5.1 durchgeführt, welches die Kosten der Implementation mit und ohne dieser Regel vergleicht. Das Experiment besteht aus 1000 Samples.

Regel	Durchschnittliche Zeit pro Token
Finales Token behalten	22.93 ms
Finales Token verwerfen	26.82 ms

Tabelle 5.4.: Kostenvergleich: Verwerfen des letzten Tokens

Das Verwerfen des letzten Tokens im Kandidaten führt zu einer Kostenerhöhung von mehr als 15 %. Eine Analyse der zugrundeliegenden Daten zeigt, dass vor allem bei kleinen Kandidaten grosse Kosten anfallen. Für einen Kandidaten der Grösse 1 sind zwei Aufrufe nötig, da der letzte verworfen wird. Dieser Overhead könnte möglicherweise reduziert werden, indem eine Minimalgrösse des Kandidaten mit einer zusätzlichen Regel vorausgesetzt wird. Aufgrund des erheblichen Kostenunterschieds wird der Ansatz nicht weiter verfolgt. Die Implementation wird geändert, damit das finale Token immer behalten wird.

Wahl von num_k

Durch num_k wird die maximale Länge eines Kandidaten festgelegt. Es ist eines von zwei hinreichenden Kriterien, um die Drafting Phase abzuschliessen. Sobald ein Kandidat die Länge num_k erreicht, wird er abgeschlossen und in die Validation gegeben, unabhängig davon, ob der Threshold erreicht wurde. Grundsätzlich funktioniert Entropy Drafting auch ohne dieses Kriterium.

Das Ziel von num_k ist es, lange Kandidaten selbst bei sehr hoher Sicherheit des Draft-Modells zu limitieren und eine Obergrenze für die Kandidatengrösse zu forcieren. num_k ist daher besonders wichtig bei lockeren (bzw. hohen) Thresholds.

Die Wahl von num_k ist abhängig von der Modellkonfiguration und den eingesetzten Thresholds. Um zu eruieren, ob sich ein Tuning lohnt, wurden Experimente mit num_k = 10 und num_k = 100 durchgeführt. In diesen hat sich gezeigt, dass die Kosten von hohen Thresholds steigen. Abgesehen davon waren die Resultate sehr ähnlich. Dies ist auch zu erwarten, da bei der Wahl von sinnvollen Hyperparametern num_k nur in Ausnahmefällen erreicht werden sollte. Da der Parameter für die Kosten der interessanten Hyperparameter-Bereiche keine Auswirkung gezeigt hat, wird er im Rahmen dieser Arbeit nicht optimiert und stattdessen arbiträr festgelegt.

5.5.4. Gegenüberstellung von Simulation und Inferenz

In diesem Kapitel werden die beiden Methoden Simulation und Inferenz rekapituliert und die wichtigsten Unterschiede aufgezeigt.

Das Ziel der Simulation ist eine möglichst günstige und genaue Approximation des Ansatzes, ohne dass eine Implementation vorausgesetzt wird. Anhand einer statischen Abfolge an Entropien wird das Verhalten simuliert. Durch die niedrigen Kosten können Breitensuchen vorgenommen werden, um vielversprechende Hyperparameter-Bereiche zu finden. Diese Bereiche können anschliessend mit der Inferenz getestet werden. Die Inferenz setzt eine Implementation vor. Durch die hohen Kosten der Generierung werden mit der Inferenz deutlich kleinere Samples verarbeitet. Eine Mindestanzahl von 10 Samples ist ratsam, da es sonst eine hohe Variabilität gibt. Beide Methoden können durch Angabe von den Modellkosten t_t und t_d ausgewertet werden. Damit können auch hypothetische Modellgrössen abgeschätzt werden. Bei der Inferenz ist es zudem möglich, die Messungen der Generierung auszuwerten. Dadurch kann die Performance unter realen Bedingungen ermittelt werden. Die Messung misst insbesondere auch die Performance der Implementation.

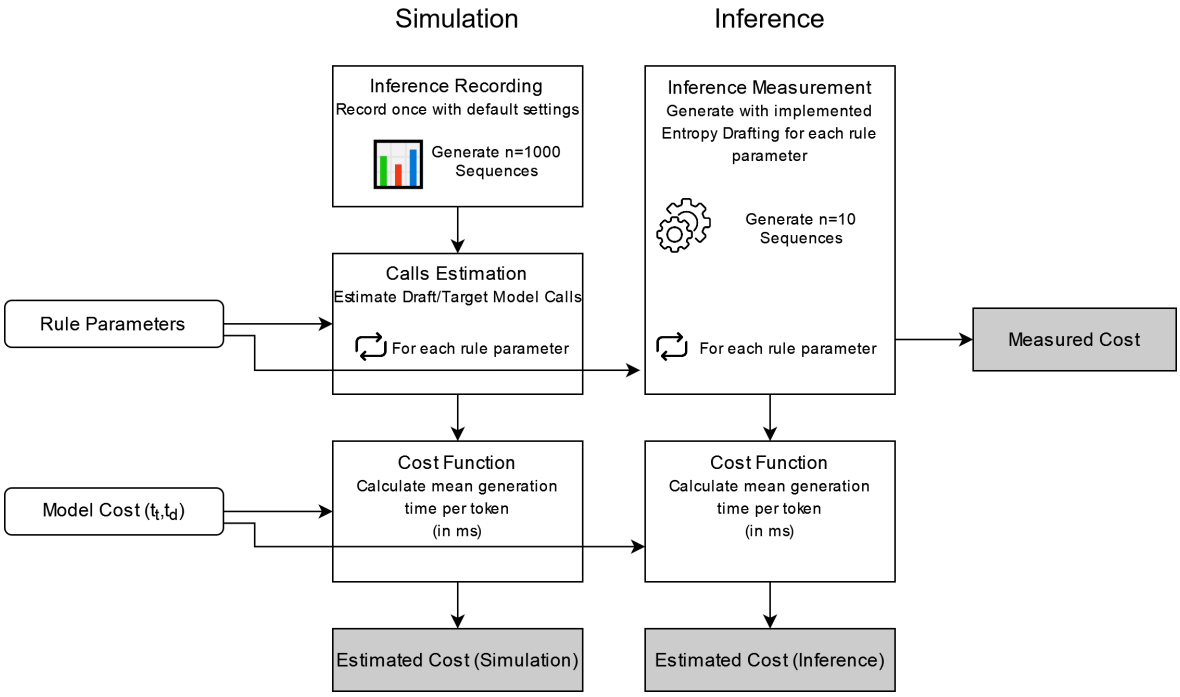


Abbildung 5.8.: Gegenüberstellung Ablauf von Simulation und Inferenz

Im Optimalfall decken sich die Resultate von Simulation und Inferenz. Es kann aber auch zu Abweichungen kommen.

6. Evaluation

6.1. Resultate (Default)

In diesem Kapitel werden die Resultate der Hyperparameter Suche diskutiert. Die Ergebnisse der Simulation werden denen der Inferenz gegenübergestellt. Es wird beurteilt, ob die Simulation die Messwerte der Inferenz zuverlässig approximieren kann. Zudem werden die optimalen Hyperparameter festgestellt.

Für die Simulation wird eine Aufzeichnung mit 1000 Samples ausgewertet. Für die Inferenz wird jede Hyperparameter-Konfiguration über 10 Samples berechnet. In beiden Fällen wird der Durchschnitt pro Token berechnet. Bedingt durch die Stichprobengrösse weist die Inferenz eine höhere Varianz auf als die Simulation.

Dieses Kapitel beschreibt die Ergebnisse vom Default-Experiment. Die Einstellungen finden sich im Versuchsaufbau 5.5.1. Die Resultate werden nach den drei Threshold Strategien gegliedert.

Für die Standard-Strategie (HF +2/-1) 2.2 betragen die durchschnittlichen Kosten 24.85 ms pro Token. Dieser Wert wird nachfolgend als Baseline für den Vergleich der Entropy Drafting Strategien verwendet.

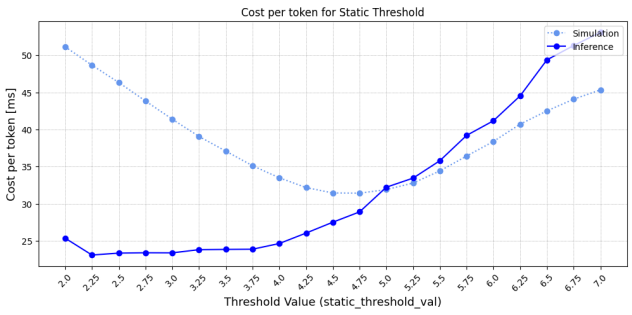


Abbildung 6.1.: Hyperparameter-Kosten Übersicht für Static Threshold

Das Diagramm 6.1 visualisiert die Kosten (y-Achse) von statischen Thresholds im Bereich 2.0 bis 7.0 für die Simulation und die Inferenz. Dies ist die Einfachste der drei Strategien. Die Kosten für den statischen Threshold wurden von der Simulation grundsätzlich zu hoch eingeschätzt. Der optimale Threshold der Simulation liegt bei einer Entropie von 4.75, was im Betracht auf das Entropie-Histogramm 5.4 realistisch erscheint. Das lokale Minimum der Inferenz liegt hingegen bei 2.25. Hierbei wurden Kosten von 23.11 ms gemessen. Damit generiert Entropy Drafting schneller als die Baseline Strategie.

Die Messungen divergieren vor allem im niedrigen Bereich deutlich von der Simulation. Während die Simulation steigende Kosten für niedrige Thresholds voraussagt, sinken sie in der Inferenz deutlich. Die Unterschiede zwischen Simulation und Inferenz werden im Kapitel Unterschiede zwischen Simulation und Inferenz 6.4.3 ausgewertet.

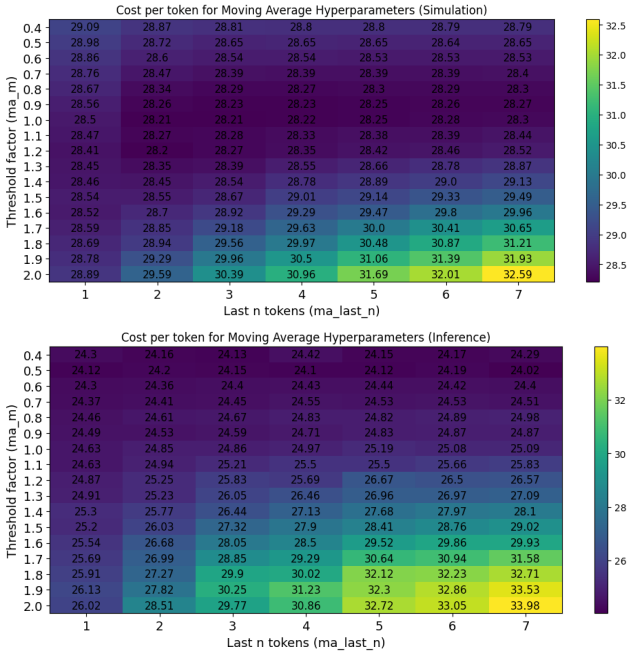


Abbildung 6.2.: Hyperparameter-Kosten Übersicht für Moving Average

Der Moving Average (MA) hat keinen statischen Grenzwert und fokussiert sich auf den relativen Trend des Entropieverlaufs. Beim MA ist der Fehler zwischen Simulation und Inferenz kleiner. Es zeigt sich zudem eine Tendenz hin zu niedrigen Threshold-Faktoren. Faktoren unter 1.0 haben zur Konsequenz, dass der Threshold selbst bei einer sinkenden Entropie erreicht wird. Die niedrigen Kosten für Threshold-Faktoren unter 1.0 sind unerwartet. Die Strategie möchte die Drafting Phasen selbst dann beenden, wenn die Entropien stark sinken, was grundsätzlich für sichere Voraussagen spricht. Die optimalen Parameter für die Inferenz sind (0.5, 7) für (*ma_m, ma_last_n*) und führen zu Kosten von 24.02 ms. Auch dies ist leicht besser als die Baseline, der Unterschied ist jedoch eher bescheiden.

Der MA ist die erste Strategie, die vergangene Entropien einbezieht. Damit soll geprüft werden, ob der Einbezug von vergangenen Tokens überhaupt hilfreich ist. Es werden nur vergangene Tokens aus der aktuellen Drafting Phase ausgewertet. Damit wird verhindert, dass Tokens mehrfach ausgewertet werden. Diese Designentscheidung äussert sich auch in der Auswertung. Für niedrige Threshold Faktoren sind die Kosten für alle *ma_last_n* praktisch gleich. Es spielt keine grosse Rolle, ob 1 oder 7 vergangene Entropien einbezogen werden. Weil die Kandidaten durch den niedrigen Faktor minimal gehalten werden, gibt es in beiden Fällen keine oder sehr wenige vergangene Tokens. Selbst bei höheren Faktoren führt der Einbezug von vergangenen Tokens zu einer Erhöhung der Kosten.

Die Auswertung legt nahe, dass die Kernideen des MA als Abbruchkriterium nicht effektiv sind. Im Hinblick auf die besseren Ergebnisse des Static Threshold scheint eine Auswertung der relativen Änderung anstelle der Absolutwerte keine Vorteile zu bringen. Andererseits hat der Einbezug von mehr vergangenen Tokens zum gleichen oder einem schlechteren Resultat geführt.

Der Cumulative Threshold ist ein statischer Grenzwert. Im Unterschied zum Static Threshold werden *cum_last_n* Tokens auch berücksichtigt. Im Gegensatz zum MA erwirkt der Einbezug von vergangenen Tokens hier tiefere Kosten. Eine Hyperparameter-Einstellung von (45, 7) für (cum_threshold_val, cum_last_n) führt in der Simulation zu minimalen Kosten. Wie schon bei den anderen Strategien tendiert die Inferenz zu einem tieferen Threshold. Bei der Inferenz betragen die Kosten unter den besten Parametern 23.52 ms. Ein bemerkenswerter Unterschied zwischen der Simulation und Inferenz sind die Kosten für niedrige Thresholds. Wie auch schon beim Static Threshold werden hohe Kosten vorausgesagt, wobei die Kosten in Inferenz tatsächlich sinken. Da die Kosten für Inferenz auch an der Diagrammgrenze noch zu sinken scheinen, wurden auch Thresholds unter 10 ausgewertet. Diese unterscheiden sich aber nicht signifikant von den aktuellen Minimalwerten.

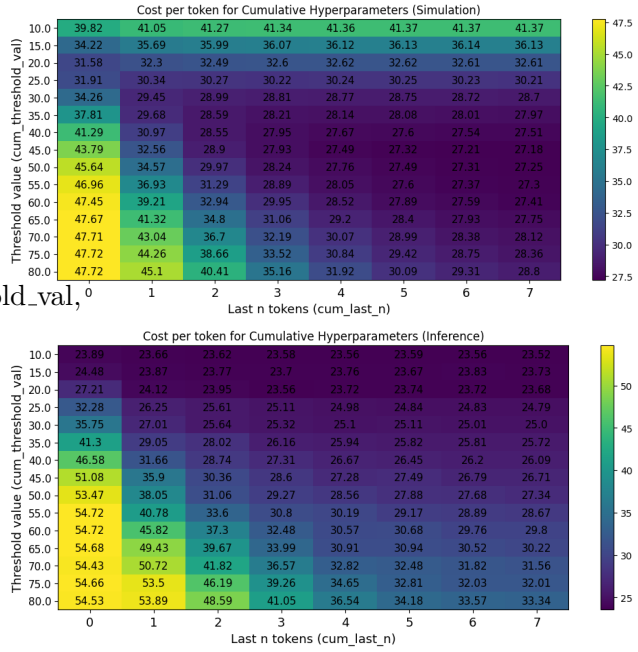


Abbildung 6.3.: Hyperparameter-Kosten Übersicht für Cumulative Threshold

Strategie	Hyperparam* Sim.	Kosten	Hyperparam* Inf.	Kosten
Static	4.75	31.43 ms	2.25	23.11 ms
Moving Avg.	(1.2, 2)	28.20 ms	(0.5, 7)	24.02 ms
Cumulative	(45, 7)	27.18 ms	(10, 7)	23.52 ms
Baseline				24.85 ms

Tabelle 6.1.: Übersicht Experimente

Zusammenfassend lässt sich sagen, dass bei allen Threshold Strategien Kostenvorteile im Vergleich zu der Baseline (+2/-1 Strategie) gemessen werden konnten. Die Resultate demonstrieren, dass Entropy Drafting nicht nur in Theorie, sondern auch praktisch funktioniert und die Inferenz messbar beschleunigen kann. Die Vorteile halten sich für dieses Experiment in einem bescheidenen Rahmen. Dies könnte damit erklärt werden, dass dieser Versuchsaufbau nicht optimale Voraussetzungen für den Ansatz bietet. Es gibt folgende Vermutungen.

- Modellgrösse: Die im Experiment verwendeten Modellgrössen sind zu klein, um effektiv von Assisted Generation und Entropy Drafting zu profitieren. Um das Potenzial auszureizen, könnten grössere (evtl. verteilte) Modelle notwendig sein.
- Input-Länge: Die Input-Länge und die Anzahl der zu generierenden Tokens pro Sample ist zu tief, um von diesem Ansatz zu profitieren. Assisted Generation und Entropy Drafting profitieren von Kontext, der sich durch einen grossen Input ergibt.
- Processing Overhead: Die für Entropy Drafting notwendigen Berechnungskosten in jeder Iteration heben einen Teil der Zeitersparnisse wieder auf.

6.2. Erhöhung der Modellgrösse

Die Wahl des Target- und Draft-Modells hat erheblichen Einfluss auf die Effektivität von Assisted Generation. Für optimale Performance wird der Betrieb von zwei möglichst unterschiedlich grossen Modellen motiviert (Gante, 2023). In Leviathan et al. (2023) wurde als Target-Modell T5-XXL (11B) verwendet. In C. Chen et al. (2023) wird für Target ein Chinchilla Modell mit 70B Parametern verwendet.

Wie im Versuchsaufbau 5.5.1 beschrieben, wurden die Experimente mit OPT-2.7B als Target-Modell vorgenommen. Die Berechnungszeiten t_t können für OPT 125M, 350M, 1.3B und 2.7B gemessen und ausgewertet werden. Diese Modelle sind deutlich kleiner als in verwandten Arbeiten. Zur Auswertung von grösseren Modellen muss entweder die Präzision verringert oder der Grafikspeicher erhöht werden.

Kann t_t für grössere Varianten extrapoliert werden? Grundsätzlich skaliert die Latenz sublinear mit der Modellgrösse. Die Latenz von Llama2-70B ist beispielsweise ungefähr 2x so hoch wie bei Llama2-13B (Agarwal et al., 2023).

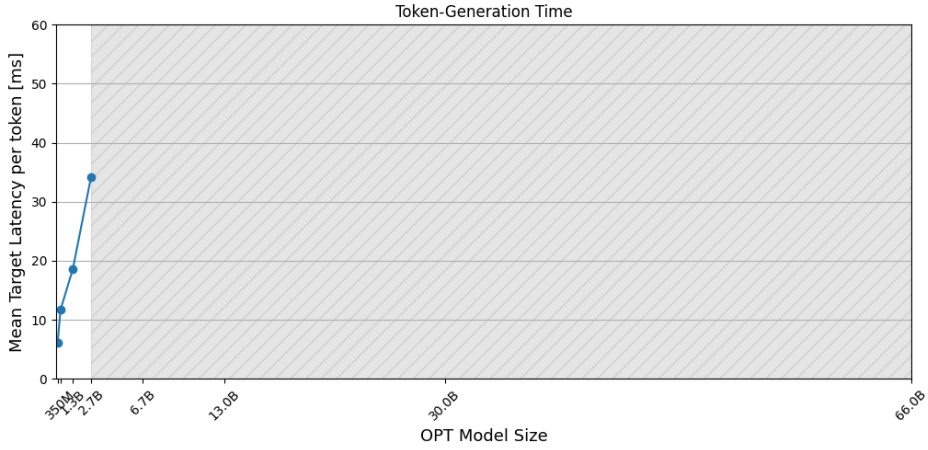


Abbildung 6.4.: t_t für OPT Modellgrössen bis OPT-66B

Eine grundsätzliche Schwierigkeit bei einer Extrapolation ist, dass die Abstände der Modellgrössen deutlich zunehmen. Wie in 6.4 dargestellt, ist der messbare Bereich bescheiden. Eine zweite Schwierigkeit ist, dass grössere Modelle meist auf anderen Hardwarekonfigurationen eingesetzt werden. Für die Inferenz von grossen Modellen werden meist verteilte Systeme eingesetzt. In Dettmers et al. (2022) wird für die Inferenz von OPT-175B (in 16-Bit Präzision) ein Cluster aus 8x NVIDIA A100 (80 GB) verwendet. Es ist nicht realistisch, mit Messungen einer Single-GPU Konfiguration, Inferenz in einem verteilten System zu approximieren.

Da Abschätzungen aufgrund der aufgeführten Gründe mit hohen Unsicherheiten verbunden sind, werden die realen Kosten des Modells gemessen. Dafür wird eine NVIDIA A40 (48 GB) On-Demand bei einem Cloud-Anbieter gemietet, um die Kosten t_t und t_d bei grösseren OPT-Modellen zu ermitteln. Diese Daten können für Simulationen verwendet werden. Zusätzlich wird auch Inferenz ausgeführt, um die Simulation mit den Messwerten zu vergleichen.

Für die Gegenüberstellung der Token-Generation Zeit zwischen beider Grafikkarten wird das Verhältnis verwendet, da t_d variiert. Grundsätzlich steigt mit dem Verhältnis 6.5 die Effektivität von Assisted Generation. Die Messungen zeigen, dass dieser Wert stark von der verwendeten Hardware abhängig ist. Das Verhältnis steigt für OPT-6.7B, allerdings nicht so stark wie angenommen. Die Messdaten liegen im Appendix A.1 bei.

Kategorie	Einstellung	Wert
Generierungsparameter	GPU	NVIDIA A40 (48GB)
	Target Modell	OPT-6.7B
	Draft Modell	OPT-125M
	Präzision	FP32

Tabelle 6.2.: Aktualisierte Systemparameter

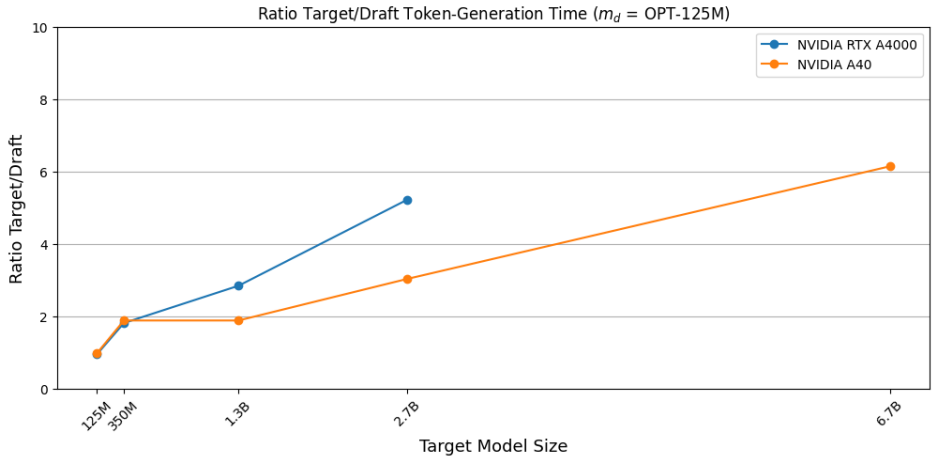


Abbildung 6.5.: Verhältnis Token-Generation Time t_t/t_d

Da t_d und t_t höher sind als bei OPT-2.7B, steigen alle Kosten. Die Simulation mit den Messwerten für OPT-6.7B ($t_d = 8$ms, $t_t = 51$ms) führt bei allen drei Threshold Strategien zu höheren Kosten bei niedrigen Thresholds. Dies ergibt Sinn, da Calls auf das Target Modell nun teurer sind und die Simulation motiviert ist, diese zu reduzieren. Dadurch steigen die Kosten für niedrige Thresholds. Wie schon im Kapitel Resultate 6.1 festgestellt, tendiert die Inferenz auf kleine Thresholds. Mit OPT-6.7B verschiebt sich das lokale Minimum allerdings auf höhere Thresholds. Der optimale Static Threshold steigt von 2.25 auf 3.75.

Strategie	Hyperparam* Sim.	Kosten	Hyperparam* Inf.	Kosten
Static	4.75	42.78 ms	3.75	30.57 ms
Moving Avg.	(1.2, 2)	39.31 ms	(0.4, 3)	32.58 ms
Cumulative	(50, 7)	37.49 ms	(15, 1)	30.05 ms
Baseline				31.56 ms

Tabelle 6.3.: Übersicht Experimente OPT-6.7B

Aus den Ergebnissen geht hervor, dass die Inferenz mit Ausnahme von MA zu höheren Thresholds tendiert. Der Cumulative Threshold erzielt das beste Ergebnis. Die Vorteile von Entropy Drafting zu der Baseline liegen in einem ähnlichen Rahmen wie bei OPT-2.7B. Die ausführlichen Ergebnisse für alle Hyperparameter-Settings liegen im Anhang A.1 bei.

6.3. Erhöhung der Input- und Ausgabelänge

Für die Experimente wurden kurze Input- und Ausgabelängen verwendet. Es besteht die Vermutung, dass die Effektivität von Assisted Generation mit längeren Input- und Ausgabelängen steigt. Die bisherige Einstellung max_new_tokens = 25 eignet sich gut für Visualisierungen, ist aber für die meisten Anwendungen zu tief angesetzt.

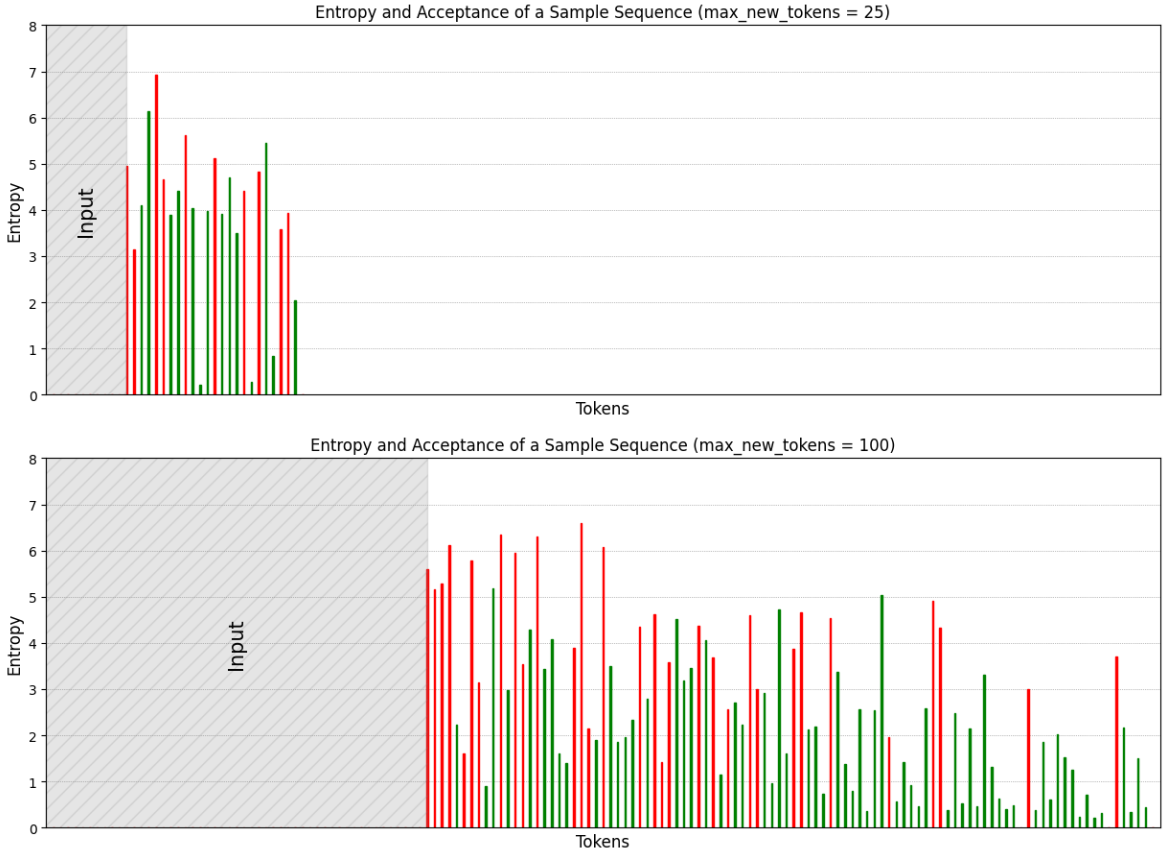


Abbildung 6.6.: Entropie und Akzeptanz über unterschiedliche Längen

In Abbildung 6.6 wird illustriert, welche Auswirkungen eine höhere Input- und Ausgabelänge auf die Entropie und Akzeptanz einer Sequenz hat. Die Farbcodierung der Balken gibt an, ob der Token akzeptiert wurde (grün) oder nicht (rot). Der obere Plot repräsentiert die aktuellen Einstellungen. Im unteren Plot wurde die Input- und Ausgabelänge um den Faktor 4 vergrössert. Alle anderen Einstellungen sowie der Input-Index wurden beibehalten. Aus dem unteren Plot geht hervor, dass die Akzeptanz zu Beginn der Generierung besonders tief ist. Im Verlauf der Sequenz gewinnt das Modell an Sicherheit, was sich in der sinkenden Entropie widerspiegelt. Dies geht einher mit einer höheren Akzeptanz.

Eine naheliegende Erklärung für die sinkende Entropie ist, dass das Modell über den Verlauf der Sequenz durch den ansteigenden Informationskontext bessere Voraussagen machen kann. Durch einen längeren Input wird der Themenbereich stärker eingegrenzt. Dies vereinfacht auch die Generierung von N-Grammen, die bereits in der Vergangenheit aufgetreten sind.

Es werden zwei Experimente durchgeführt, um die Performance mit höheren Längen zu messen.

Experiment	Einstellung	Wert
1	Input Tokens	25
	max_new_tokens	100
2	Input Tokens	100
	max_new_tokens	1000

Tabelle 6.4.: Einstellungen für höhere Längen

Experiment 2 hat einen 10x höheren Wert für max_new_tokens als Experiment 1. max_new_tokens ist ein Maximalwert, das Modell kann die Generierung schon früher mit einem End-Token beenden. Aus den Aufzeichnungen geht hervor, dass dies vor allem bei grossen Werten für max_new_tokens passiert und einen erheblichen Einfluss auf die Ausgabelänge hat. Bei Experiment 1 werden durchschnittlich 96 von 100 Tokens generiert, bei Experiment 2 sind es 515 von 1000. Für die Bestimmung der Durchschnittszeiten werden die effektiven Ausgabelängen verwendet.

Exp.	Strategie	Hyperparam* Sim.	Kosten	Hyperparam* Inf.	Kosten
1	Static	4.00	31.12 ms	2.25	20.95 ms
	Moving Avg.	(1.0, 4)	27.04 ms	(0.4, 1)	21.73 ms
	Cumulative	(40, 7)	26.71 ms	(15, 3)	20.82 ms
	Baseline				21.70 ms
2	Static	2.75	31.46 ms	2.00	20.63 ms
	Moving Avg.	(1.1, 5)	25.20 ms	(0.5, 1)	21.56 ms
	Cumulative	(15, 7)	29.02 ms	(10, 7)	20.74 ms
	Baseline				21.71 ms

Tabelle 6.5.: Übersicht Experimente Input- und Ausgabelänge

Es zeigen sich keine wesentlichen Unterschiede zwischen den beiden Experimenten. Die besten Ergebnisse werden weiterhin mit dem Static und Cumulative Threshold erzielt. Der Kostenvorteil hält sich in einem ähnlichen Rahmen wie bei den vorherigen Experimenten. Die Kosten sind für alle Strategien gesunken während die Baseline in einem ähnlichen Verhältnis mit den Entropy Drafting Strategien steht wie vorher.

Die ausführlichen Ergebnisse für alle Hyperparameter-Settings liegen in Anhang A.2 und A.3 bei.

6.4. Laufzeit- und Fehleranalyse

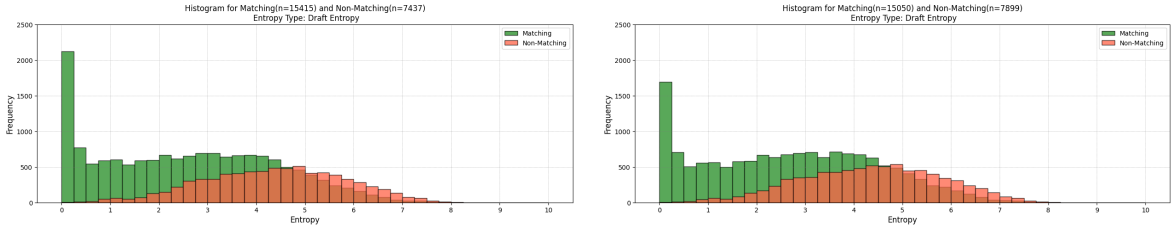
6.4.1. Auswirkungen von Repetitionen

Bei den ersten Auswertungen dieser Arbeit kam es in einigen Fällen zu Repetitionen im Output der Generierung. Das Phänomen ist daran zu erkennen, dass sich N-Gramme oftmals endlos repetieren. In Holtzman et al. (2020) wird eine Repetition als eine Sequenz definiert, die aus mindestens zwei Tokens besteht und sich am Ende der Textgenerierung mindestens dreimal wiederholt. Das Ausmass an Repetitionen ist stark abhängig von der verwendeten Decoding Strategie. Am meisten treten sie bei deterministischen Strategien wie Greedy Decoding auf (Holtzman et al., 2020). Endlose Repetitionen sind unerwünscht, da sie zu einem sinnlosen Output führen. Bei den Experimenten hat sich gezeigt, dass sich Repetitionen stark auf die Entropie und die Akzeptanz auswirken. Wird eine Sequenz in der Generierung wiederholt, sinkt die Entropie sehr stark. Durch Repetitionen entstehen in den Samples Bereiche mit Entropien nahe zu 0 mit beinahe perfekter Akzeptanz.

Diese Eigenschaften könnten durch Assisted Generation ausgenutzt werden, indem sehr grosse Kandidaten über die Repetition gebildet werden. Mit einer von Entropie abhängigen Strategie könnten Kandidaten mit einer Länge von hunderten Tokens erstellt werden, welche aufgrund des deterministischen Charakters dieser Wiederholungen fast sicher angenommen werden würden. Dies würde sich positiv auf die Performance auswirken, dabei aber die Zweckmässigkeit untergraben. Da Repetitionen unerwünschte Phänomene sind, ergibt es keinen Sinn, ihr Vorkommen zwecks besserer Performance auszunutzen. Aus diesem Hintergrund werden Repetitionen mit dem Parameter no_repeat_ngram_size in allen Experimenten verhindert.

Die Einschränkung von Repetitionen passierte erst nach der Eruierung von der Effektivität von Entropie als Abbruchkriterium 5.2. Daher entstand der Verdacht, dass die ursprüngliche Auswertung aufgrund Auftreten von Repetitionen positiv verzerrt wurde und daraus das Potenzial überschätzt wurde. Um dies auszuschliessen, wurde dasselbe Histogramm mit und ohne Repetitionen generiert. (Abbildung 6.7) Bei Entropien nahe null zeigt sich, dass beim Sample mit Repetitionen etwa 25 % mehr akzeptierte Werte vorkommen. Allerdings ändern sich die Grunderkenntnisse aus dem Histogramm nicht.

Damit kann ausgeschlossen werden, dass die Einschränkung von Repetitionen hinsichtlich der Resultate relevant ist.



(a) Sample mit Repetitionen

(b) Sample ohne Repetitionen

Abbildung 6.7.: Vergleich Histogramm Entropie mit/ohne Repetitionen

6.4.2. Kandidatenlänge über Drafting Iterationen

Eine Auswertung der Kandidatenlänge über die verschiedenen Strategien ermöglicht es, Erkenntnisse über die Threshold Strategien und ihre Adaptivität zu sammeln. Für diese Auswertung besonders interessant ist der Vergleich zwischen den Entropy Drafting Strategien mit der Baseline Strategie. Für diese Auswertung wurde mit max_new_tokens = 100 generiert und ein Sample ausgewählt, das sowohl einfache als auch schwierige Stellen enthält. Es wurden dabei die gleichen Hyperparameter wie bei Experiment 1 von Erhöhung der Input- und Ausgabelänge 6.3 verwendet.

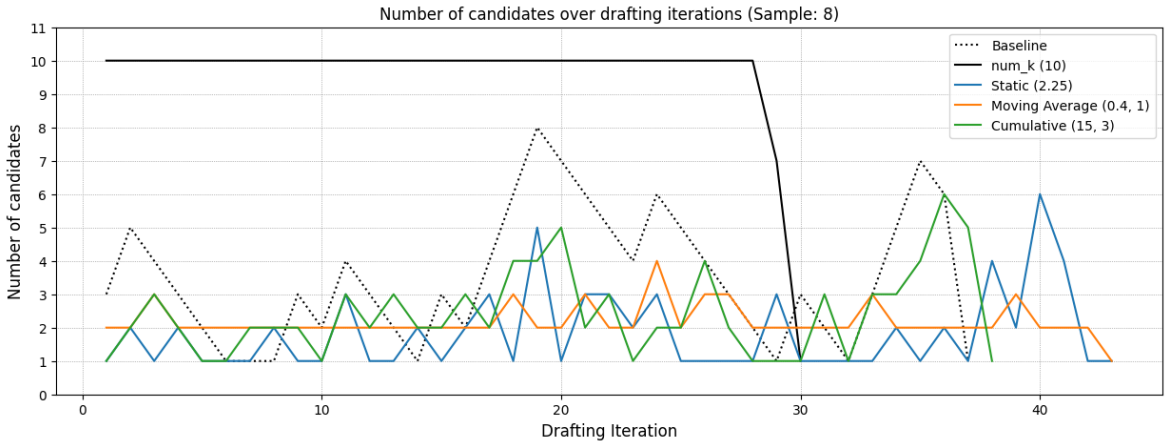


Abbildung 6.8.: Kandidatenlänge über Drafting Iterationen für ein Sample

Als Referenz wurde mit einer statischen Strategie mit num_k = 10 generiert. Zu Ende der Generierung reduziert sich die Kandidatenlänge auf die Anzahl der verbleibenden Tokens. Bei der Baseline Strategie sind die Schrittinkremente von +2/-1 gut an den Steigungen erkennbar. Eine Auffälligkeit ist, dass die Baseline Strategie nicht bei dem initialisierten Wert von 5 startet. Der CandidateGenerator wird nur zu Beginn und nicht für jedes Sample neu initialisiert. Daher verwendet dieses Sample zu Beginn das num_k, mit welchem das vorherige Sample abgeschlossen hat. Weiterhin wird die Kandidaten-Mindestlänge von 1 nie unterschritten.

Grundsätzlich verhalten sich die Entropy Drafting Strategien zurückhaltend. Dies ist bedingt durch die tiefen Threshold Hyperparameter, die eine Bildung von grossen Kandidaten erschweren.

Zu Beginn der Generierung (Iteration 5) sinken die Kandidatenlängen, was auf einen schwierigen Abschnitt hindeutet. Die Kandidatenlänge sinkt bei allen Strategien. Zwischen Iteration 15 und 20 steigen die Kandidatenlängen für die Baseline und den Cumulative Threshold an. Der Moving Average verändert sich kaum und der static Threshold ist eher volatil. Die abrupten Änderungen für den static Threshold könnten durch den fehlenden Kontext erklärt werden. Der static Threshold entscheidet nur anhand des aktuellen Tokens.

Die Auswertung gibt ausserdem eine plausible Erklärung für die durchgängig höheren Kosten der Strategie Moving Average. Der MA benötigt definitionsgemäss immer mindestens einen vergangenen Token für die Auswertung der Threshold Strategie. Die Mindestkandidatengrösse steigt dadurch auf 2. Während die anderen Strategien in schwierigen Abschnitten die Kandidatengrösse auf 1 reduzieren können, generiert der MA auch in diesen Fällen mindestens 2 Kandidaten und verschwendet damit potenziell Aufrufe.

6.4.3. Unterschiede zwischen Simulation und Inferenz

Das Ziel der Simulation ist es, die Inferenz möglichst gut zu approximieren. Im Idealfall decken sich die Ergebnisse. Bei den Experimenten hat sich allerdings gezeigt, dass es starke Abweichungen bei tiefen Thresholds gibt. Während die Simulation bei allen Strategien hohe Kosten für tiefe Thresholds voraussagt, sind diese in Inferenz tiefer.

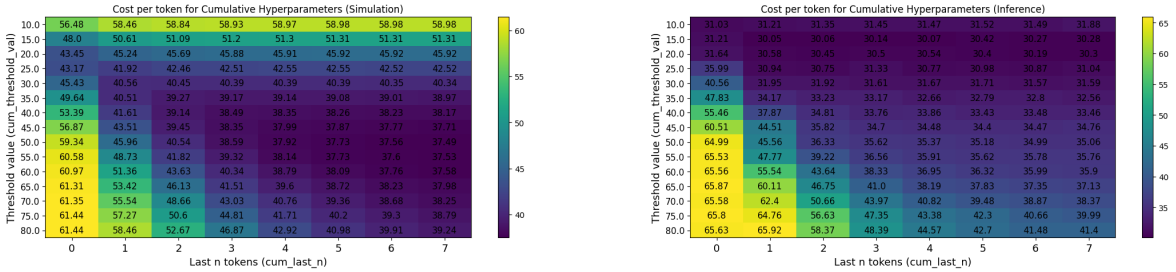


Abbildung 6.9.: Vergleich Simulation und Inferenz anhand Experiment: Erhöhung der Modellgrösse 6.2 für Cumulative Threshold

Die durchgehende Präferenz von tiefen Thresholds signalisiert, dass Entropy Drafting in den Experimenten auf Schwierigkeiten stiess. Dafür gibt es mehrere Theorien.

- Modellgrösse Draft: Als Draft-Modell wurde ein OPT Modell mit 125 Millionen Parametern verwendet. Möglicherweise könnte sich die Verwendung des nächstgrösseren Modells (OPT-350M) lohnen. Dies lohnt sich nur, wenn die Vorteile einer höheren Draft Akzeptanz die Nachteile von höheren Draft-Kosten übersteigen.
- Schwieriger Task: Der Datensatz C4 besteht aus Crawling-Daten. Die Qualität unter den Samples variiert stark. Einige Samples bestehen aus Zeitungstexten von guter Qualität während andere eher kryptisch und zusammenhangslos wirken. Letztere könnten die Sicherheit von Text Generation gesenkt haben. Gegen diese These spricht, dass OPT selbst mit Crawling Daten trainiert wurde (Zhang et al., 2022) und angenommen wird, dass verschiedene Modellgrössen sich ähnlich verhalten.
- Unterschied Sample Size: Für die Simulation wurde der Durchschnitt aus 1000 Samples berechnet, für die Inferenz nur aus 10. Aufgrund der kleinen Sample-Grösse könnte es sein, dass die Samples homogen sind und wenige Fälle enthalten, bei denen der Ansatz Vorteile bringt.
- Kostenunterschied Draft und Target: Möglicherweise ist der Kostenunterschied zwischen Draft und Target zu wenig gross. Bei der Erhöhung der Modellgrösse 6.2 sind die Kosten für niedrige Thresholds leicht angestiegen. Der Einsatz von noch grösseren (und teureren) Target-Modellen sollte die Effektivität fördern.

6.4.4. Effizienz der Implementation

Dieses Kapitel beschäftigt sich mit der Effizienz der Entropy Drafting Implementation dieser Arbeit. Im Gegensatz zu der Baseline Strategie wird bei Entropy Drafting nach jedem generierten Draft Token überprüft, ob der Threshold überschritten wurde. Dies führt zu zusätzlichen Rechenoperationen zwischen den generierten Drafting Tokens.

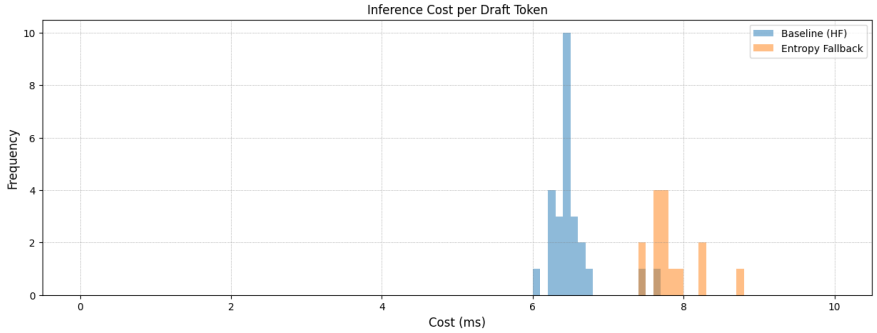


Abbildung 6.10.: Histogramm der durchschnittlichen Draft Kosten für Baseline und Entropy Drafting (NVIDIA A4000, m_d = OPT-125M)

Das Histogramm 6.10 vergleicht die Kosten pro Draft Token für Baseline und Entropy Drafting. Die Kosten von Entropy Drafting liegen etwa 1.5ms über der Baseline. Dieser Unterschied konnte auch in anderen Samples gemessen werden. Die Implementation hat somit einen Overhead von ungefähr 20 %, die für die Berechnung von Entropy Drafting aufkommen. Dieser Overhead ist bedeutend, da sich die Mehrkosten mit jedem Drafting Call akkumulieren.

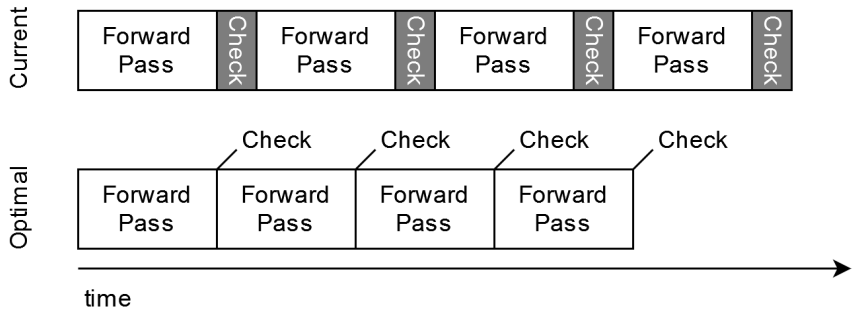


Abbildung 6.11.: Vergleich Drafting Effizienz von aktueller mit optimaler Implementation

Bei der Implementation wurde nur auf funktionale Aspekte geachtet, es wurde keine Optimierung vorgenommen. Da die für Entropy Drafting notwendigen Rechenoperation im Vergleich zu den Forward-Passes vernachlässigbar sind, spricht nichts gegen eine effizientere Implementation. Eine effiziente Implementierung dieses Ansatzes könnte ein interessantes Folgeprojekt dieser Forschungsarbeit sein und würde die Idee für den produktiven Gebrauch erschliessen.

6.5. Fazit der Evaluation

Der Zweck dieser Experimente ist es, die optimalen Hyperparameter für die Strategien zu finden und zu eruieren, ob Entropy Drafting Vorteile im Vergleich zu der Baseline Strategie bringt. Zusätzlich zu der Hugging Face Baseline werden auch die Kosten für Generierung ohne Assistenzmodell ermittelt. Das Balkendiagramm 6.12 zeigt die Kosten für alle Strategien, nach den Experimenten gruppiert.

Im Hauptexperiment hat Entropy Drafting mit allen Strategien zu einer Beschleunigung der Generierung geführt. Es zeigte sich, dass die Inferenz unerwartet niedrige Thresholds präferiert. Möglichen Gründe dafür wurden in Unterschiede zwischen Simulation und Inferenz 6.4.3 behandelt.

Um zu untersuchen, ob eine Erhöhung der Modellgrösse oder der Input-/Ausgabelänge die Vorteile weiter erhöht, wurden Anschlussexperimente durchgeführt. Die Erhöhung der Modellgrösse hat zu interessanten Erkenntnissen über das Kostenverhalten geführt. Im Vergleich zu der Hugging Face Baseline waren die Kostenvorteile der Strategien ähnlich wie beim Hauptexperiment.

Die Erhöhung der Modellgrösse führt zu einem substanziellen Anstieg der Kosten für Generierung ohne Assistenzmodell. Die Kosten können bei OPT-6.7B mit einem Assistenzmodell beinahe halbiert werden. Dies ist bemerkenswert, wenn man bedenkt, dass das Modell nur von 2.7B auf 6.7B Parameter erhöht wurde und verdeutlicht, dass Assisted Generation für grosse Modellunterschiede besser funktioniert. OPT gibt es bis zu einer Grösse von 175B Parametern (Zhang et al., 2022). Es ist zu erwarten, dass die Vorteile für ein solches Modell noch viel höher liegen.

Die Generierung hat sich im Vergleich mit der bisherigen Hugging Face Implementation bis zu 1.07x beschleunigt. Im Vergleich zu Generierung ohne Assistenzmodell wurde bis zu 1.89x schneller generiert. Eine tabellarische Gesamtübersicht der Experimente mit Angabe der optimalen Hyperparameter findet sich in Tabelle 6.6.

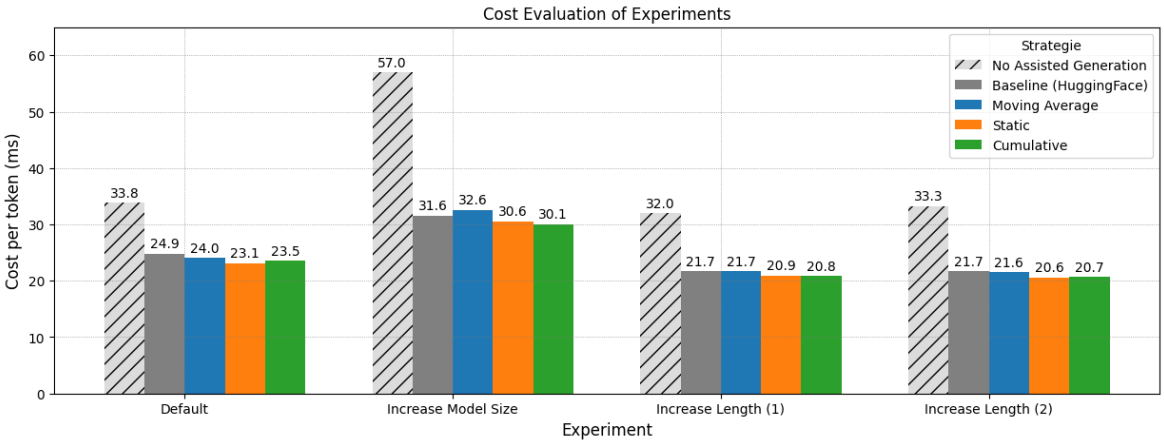


Abbildung 6.12.: Kostenauswertung über alle Experimente

Experiment	Strategie	Hyperparameter*	Kosten
Default 6.1	Static	2.25	23.11 ms
	Moving Average	(0.5, 7)	24.02 ms
	Cumulative	(10, 7)	23.52 ms
	Baseline (HF)		24.85 ms
	Kein Assistenzmodell		33.84 ms
Erhöhung Modellgrösse 6.2	Static	3.75	30.57 ms
	Moving Average	(0.4, 3)	32.58 ms
	Cumulative	(15, 1)	30.05 ms
	Baseline (HF)		31.56 ms
	Kein Assistenzmodell		57.03 ms
Erhöhung Länge (1) 6.3	Static	2.25	20.95 ms
	Moving Average	(0.4, 1)	21.73 ms
	Cumulative	(15, 3)	20.82 ms
	Baseline (HF)		21.70 ms
	Kein Assistenzmodell		32.05 ms
Erhöhung Länge (2) 6.3	Static	2.00	20.63 ms
	Moving Average	(0.5, 1)	21.56 ms
	Cumulative	(10, 7)	20.74 ms
	Baseline (HF)		21.71 ms
	Kein Assistenzmodell		33.27 ms

Tabelle 6.6.: Übersicht Experimente Inferenz mit den besten Hyperparametern

7. Konklusion

Die vorliegende Arbeit hat untersucht, ob Speculative Decoding mit einem entropiebasierten Ansatz verbessert werden kann. Als Basis für Experimente und Implementation wurde die Transformers Library von Hugging Face verwendet. Es wurden drei Strategien entwickelt, die aufgrund der Drafting-Entropie bestimmen, wie viele Tokens generiert werden sollen. Dieser Ansatz wird Entropy Drafting genannt.

Durch Experimente in verschiedenen Konfigurationen konnte nachgewiesen werden, dass Entropy Drafting die Generierung im Vergleich zu der bisherigen Hugging Face Implementation von Assisted Generation um bis zu 1.07x beschleunigt. Im Vergleich zu herkömmlicher Generierung ohne Assistenzmodell wurde bis 1.89x schneller generiert.

Die Ergebnisse dieser Arbeit sind im Wesentlichen Analysen und Erkenntnisse, die entscheidend für Designentscheidungen waren (z.B. Verwerfen des letzten Tokens). Weiterhin auch die optimalen Hyperparameter und die Resultate, die durch eine zweistufige Hyperparametersuche eruiert wurden. Die erarbeiteten Tools (Logging-Klassen, Auswertungen) können bei zukünftiger Forschung von Nutzen sein.

Eigene Experimente sowohl als auch Erkenntnisse aus verwandten Arbeiten legen nahe, dass dieser Ansatz für grössere Modelle viel Potenzial bietet. Es gibt mehrere Richtungen für zukünftige Forschung. Auf diese wird im Ausblick 8 näher eingegangen.

Auf Code-Snippets im Anhang wird verzichtet. Stattdessen wird ein GitHub Repository mit allen relevanten Informationen und Source Code zu Entropy Drafting im Anhang A.1 bereitgestellt.

7.1. Reflexion

Diese Bachelorarbeit war sehr spannend, fordernd, aber vor allem lehrreich. Die wöchentlichen Meetings waren essenziell für den Fortschritt dieser Arbeit. Der Austausch war eine wichtige Quelle für Feedback, Inputs und hat mir geholfen, die Konzepte besser zu verstehen.

Die erste Herausforderung war die Erweiterung der Transformers Library. Ich habe hier keine Hürden erwartet. Allerdings zeigte sich, dass die Änderung der Library nicht so einfach funktioniert wie erwartet und genug lange andauerte, um ein eigenes Kapitel in dieser Arbeit zu bekommen.

Es zeigte sich, wie wichtig eine einheitliche Definition von Begriffen ist. Dies war in dieser Arbeit besonders wichtig, weil sie auf mehreren Arbeiten aufbaut, die z.T. grosse Überschneidungen in ihren Konzepten haben. Dies fand ich während dieser Arbeit herausfordernd.

Durch Änderungen und Umstellungen mussten die Experimente der Hyperparameter Suche mehrfach ausgeführt werden. Gründe dafür waren unwesentliche Änderungen wie z.B. Änderungen an der Visualisierung, aber auch erhebliche Änderungen am Algorithmus, die zu anderen Resultaten führten. Die Wiederholungen der Experimente waren sehr zeitintensiv, da sie auf zwei unterschiedlichen Setups ausgeführt werden mussten und einige Inferenz-Experimente über 10h dauerten.

Bezüglich der Planung kam es bei dem Auswerten der Strategien zu erheblichen Verzögerungen. In den letzten 3 Wochen habe ich einen Fehler in der Berechnung der Kosten gefunden. Gleichzeitig offenbarte sich ein Defizit in der Implementation, das die Inferenz stark verlangsamte. Die Behebung mit Wiederholung von allen Experimenten kostete viel Zeit und führe zu neuen Ergebnissen. Es war rückblickend eine gute Idee, schon etwas früher mit dem Schreiben zu beginnen, damit solche weitreichenden Änderungen noch realisiert werden konnten. Gleichzeitig hat es mir (wieder einmal) vor Augen geführt, wie wichtig umfassendes und frühzeitiges Testen ist.

Es gibt noch einige Themen, die ich gerne noch ausprobiert hätte. Leider liess der zeitliche Rahmen dies nicht mehr zu. Dazu gehören z.B. Inferenz auf verteilten Systemen oder Quantisierung von grossen Modellen.

8. Ausblick

8.1. Verteilte Inferenz und grosse Modelle

Die Experimente mit dem grösseren Modell (OPT-6.7B) haben gezeigt, dass höhere Thresholds präferiert wurden. Je teurer das Target-Modell, desto attraktiver ist die Generierung von langen Kandidaten mit dem günstigen Draft-Modell. Die Auswertung im Fazit 6.12 hat gezeigt, dass die Kostenersparnisse durch den Einsatz eines Assistenzmodells bei der Erhöhung auf OPT-6.7B stark gestiegen sind.

Es ist allerdings unklar, ob Entropy Drafting lediglich durch eine Erhöhung der Modellgrösse einen deutlichen Kostenvorteil gegenüber der Hugging Face Baseline Strategie gewinnt.

Wie in der Problemstellung 2.1 festgestellt, wird die Generierung von LLMs hauptsächlich von der Speicherbandbreite limitiert (Leviathan et al., 2023). Dies ist besonders relevant für verteilte Inferenz, wenn ein Modell über mehrere Grafikkarten geladen ist und die Bandbreite zwischen den Grafikkarten der limitierende Faktor ist. Im Rahmen dieser Arbeit wurden nur Experimente auf Single-GPU Konfigurationen ausgeführt. Inferenz auf grossen Modellen in verteilten Settings wäre daher ein interessanter Bereich für Anschlussforschung.

8.2. Implementation

Wie in Effizienz der Implementation 6.4.4 beschrieben, ist die aktuelle Implementation noch nicht effizient. Für jedes generierte Draft-Token fliesst etwa 20 % der Berechnungszeit in die Entropy Threshold Logik. Diese Logik wendet die Strategien an und eruiert anhand der Entropien, ob weiter generiert werden soll oder nicht.

Die dafür notwendigen Operationen (v.a. Berechnung der Entropie(n)) sind im Verhältnis zu den Operationen eines Forward-Passes vernachlässigbar. Eine optimierte Implementation könnte die 20 % Overhead stark reduzieren und das Drafting effektiver machen. Die Kostenvorteile davon würden Entropy Drafting insbesondere von der Hugging Face Strategie weiter abheben.

8.3. Sampling

Die Experimente in dieser Arbeit limitieren sich auf Greedy Decoding. Die Experimente in Leviathan et al. (2023) und C. Chen et al. (2023) wurden mit Greedy Decoding sowohl als auch mit Speculative Sampling durchgeführt. Beide Papers verwenden für ihre Implementation von Speculative Sampling stochastische Methoden. Wie im Kapitel Decoding Strategien 2.2 thematisiert, sind stochastische Methoden vor allem für Aufgaben relevant, die open-ended sind und von Diversität und Kreativität im Output profitieren. In Shi et al. (2024) haben stochastische Decoding Methoden deutlich bessere MAUVE Scores für offene Text-Generierung erreicht im Vergleich zu deterministischen Decoding Methoden. Dies motiviert die Erweiterung und Evaluation mit Sampling.

Bei der Verwendung von Sampling statt Greedy Decoding zeigen Benchmarks durchgehend niedrigere Speed-Ups (Leviathan et al. (2023), C. Chen et al. (2023)). Wie schon in Gante (2023) festgestellt, funktioniert Speculative Decoding besser mit Greedy Decoding. Dennoch könnten Experimente mit Sampling interessante Erkenntnisse bringen.

8.4. Benchmarks

In dieser Arbeit wurde lediglich Text-Generierung evaluiert. Für eine abschliessende und umfangreiche Evaluation von Entropy Drafting sind Benchmarks notwendig. Dies ermöglicht insbesondere den Vergleich mit anderen Arbeiten.

Für Benchmarks stehen vor allem die Tasks Übersetzung und Zusammenfassung im Zentrum. In Leviathan et al. (2023) wurden dafür T5 Modelle auf Englisch–Deutsch Übersetzung (WMT EnDe) bzw. Zusammenfassung (CNN/DM) fine-tuned. In Kim et al. (2023) wurden dieselben Tasks gewählt. Für Übersetzung wurden hier die Datasets IWSLT und WMT verwendet. Für Zusammenfassung XSUM und CNN/DM. In C. Chen et al. (2023) wurde die Performance zudem für einen Code-Generation-Task mit HumanEval (M. Chen et al., 2021) ausgewertet.

Es ergibt daher Sinn, Entropy Drafting mit diesen Tasks zu evaluieren. Einerseits erlaubt das einen Vergleich der Speed-Up Rate mit anderen Arbeiten. Andererseits können die Beschleunigung für weitere Tasks, über die Text-Generierung hinaus, ermittelt werden.

8.5. Beam Search

In der Aufgabenstellung 1.2 für diese Bachelorarbeit wurden zwei Methoden vorgeschlagen. Eine davon ist das Generieren von mehreren Kandidaten mit Beam Search, um die Übereinstimmung mit dem Target-Modell zu verbessern. Beam Search ist eine deterministische Decoding-Strategie, die die wahrscheinlichste Sequenz über mehrere Schritte auswählt (Shi et al., 2024). Im Gegensatz zu Greedy Decoding werden so auch unwahrscheinliche Tokens in Erwägung gezogen, wenn die Sequenz als Ganzes eine hohe Wahrscheinlichkeit hat.

In Leviathan et al. (2023) der Einbezug von Beam Search in Speculative Decoding als interessante Richtung für zukünftige Forschung thematisiert. Beam Search kombiniert mit Speculative Decoding wirft einige neue Fragen auf, die in dieser Arbeit nicht behandelt wurden.

8.6. Modalität

Wie schon in Leviathan et al. (2023) festgestellt, wären Experimente mit Speculative Decoding in anderen Modalitäten ausser Text interessant. In der Zwischenzeit wurden Papers veröffentlicht, die Speculative Decoding in multimodalen Anwendungen evaluieren. In Gagrani et al. (2024) wird ein Image-Text Task evaluiert, wobei mit Speculative Decoding eine Text-Beschreibung für ein Bild generiert wird. Dies wird um bis zu 2.37x beschleunigt. Die Autoren stellen aber auch fest, dass ein Draft-Modell ohne die Bildinformationen (text-only) ähnliche Performance erreicht, wie eines mit Bildinformationen.

Weitere Experimente von Speculative Decoding auf multimodalen Tasks, die durch autoregressive Generierung limitiert werden, ist eine interessante Richtung. Insbesondere, weil es für multimodale Tasks noch viele Forschungsmöglichkeiten gibt.

Abbildungsverzeichnis

Tabellenverzeichnis

Akronyme

AR-LLM Auto-Regressive Large Language Models. II

FLOPS Floating point operations per second. II

FP Forward Pass. II

GB Gigabyte. II

GPU Graphics processing unit. II

HF Hugging Face. II

KD Knowledge Distillation. II

LLM Large Language Models. II

NLP Natural language processing. II

OOP Objektorientierte Programmierung. II

PEFT Parameter-Efficient Fine-Tuning. II

SSE Sum of squared errors. II

TPU Tensor Processing Unit. II

UML Unified Modeling Language. II

VRAM Video Random Access Memory. II

Glossar

closed-ended Tasks Aufgaben, bei denen das Modell von vordefinierten Möglichkeiten aussucht. (z.B. Multiple Choice) . II

Draft Modell Kleineres (und schnelleres) Modell zur Generierung von Kandidaten. II

Forward Pass Durchlauf von Input Daten durch ein neuronales Netzwerk um eine Ausgabe zu berechnen.. II

Halluzination Phänomen, bei dem unzutreffende oder erfundene Informationen generiert werden. II

Hyperparameter Konfigurationsvariable zur Steuerung von einem Algorithmus, die optimiert werden kann. II

N-Gramm Zerlegung eines Textes in N aufeinanderfolgende Fragmente (z.B. Buchstaben, Silben oder Wörter) . II

open-ended Tasks Aufgaben, bei denen das Modell offene Antworten geben kann. . II

Overhead zusätzliche Aufwendungen, die nicht direkt zum eigentlichen Resulatat beitragen. II

Target Modell Grösseres (und langsameres) Modell zur Validierung von Kandidaten. II

Threshold Grenzwert. II

Akronyme

AR-LLM Auto-Regressive Large Language Models. II

FLOPS Floating point operations per second. II

FP Forward Pass. II

GB Gigabyte. II

GPU Graphics processing unit. II

HF Hugging Face. II

KD Knowledge Distillation. II

LLM Large Language Models. II

NLP Natural language processing. II

OOP Objektorientierte Programmierung. II

PEFT Parameter-Efficient Fine-Tuning. II

SSE Sum of squared errors. II

TPU Tensor Processing Unit. II

UML Unified Modeling Language. II

VRAM Video Random Access Memory. II

Literatur

Agarwal, M., Qureshi, A., Sardana, N., Li, L., Quevedo, J., & Khudia, D. (2023, Oktober). LLM Inference Performance Engineering: Best Practices. Verfügbar 9. Mai 2024 unter https://www.databricks.com/blog/llm-inference-performance-engineering-best-practices

amyeroberts. (2024, Januar). Release v4.37 Qwen2, Phi-2, SigLIP, ViP-LLaVA, Fast2SpeechConformer, 4-bit serialization, Whisper longform generation · huggingface/transformers. Verfügbar 28. April 2024 unter https://github.com/huggingface/transformers/releases/tag/v4.37.0

Burton, F. W. (1985). Speculative computation, parallelism, and functional programming. *IEEE Transactions on Computers*, *C-34*(12), 1190–1193. https://doi.org/10.1109/TC.1985.6312218

Cai, T., Li, Y., Geng, Z., Peng, H., Lee, J. D., Chen, D., & Dao, T. (2024). Medusa: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads [_eprint: 2401.10774].

Chang, C.-C., Reitter, D., Aksitov, R., & Sung, Y.-H. (2023). KL-Divergence Guided Temperature Sampling [_eprint: 2306.01286].

Chen, C., Borgeaud, S., Irving, G., Lespiau, J.-B., Sifre, L., & Jumper, J. (2023). Accelerating Large Language Model Decoding with Speculative Sampling [_eprint: 2302.01318]. https://arxiv.org/abs/2302.01318

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., … Zaremba, W. (2021). Evaluating Large Language Models Trained on Code [_eprint: 2107.03374].

configuration_utils.py at v4.40.2. (2024, März). Verfügbar 13. Mai 2024 unter https://github.com/huggingface/transformers/blob/v4.40.2/src/transformers/generation/configuration_utils.py

Deng, Y., Zhao, N., & Huang, X. (2023). Early ChatGPT User Portrait through the Lens of Data. *2023 IEEE International Conference on Big Data (BigData)*, 4770–4775. https://doi.org/10.1109/BigData59044.2023.10386415

Dettmers, T., Lewis, M., Belkada, Y., & Zettlemoyer, L. (2022). GPT3.int8(): 8-bit Matrix Multiplication for Transformers at Scale. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho & A. Oh (Hrsg.), *Advances in Neural Information Processing Systems* (S. 30318–30332, Bd. 35). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2022/file/c3ba4962c05c49636d4c6206a97e9c8a-Paper-Conference.pdf

Esterbrook, C. (2001, April). Using Mix-ins with Python — Linux Journal. Verfügbar 26. April 2024 unter https://www.linuxjournal.com/article/4540

Fu, Y., Bailis, P., Stoica, I., & Zhang, H. (2024, Februar). Break the Sequential Dependency of LLM Inference Using Lookahead Decoding [_eprint: 2402.02057].

Gagrani, M., Goel, R., Jeon, W., Park, J., Lee, M., & Lott, C. (2024). On Speculative Decoding for Multimodal Large Language Models [_eprint: 2404.08856].

Gante, J. (2023). Assisted Generation: a new direction toward low-latency text generation. https://doi.org/10.57967/hf/0638

Gong, N., & Yao, N. (2023). A generalized decoding method for neural text generation. *Computer Speech & Language*, *81*, 101503. https://doi.org/https://doi.org/10.1016/j.csl.2023.101503

Gou, J., Yu, B., Maybank, S. J., & Tao, D. (2021). Knowledge Distillation: A Survey. *International Journal of Computer Vision*, *129*(6), 1789–1819. https://doi.org/10.1007/s11263-021-01453-z

Holtzman, A., Buys, J., Du, L., Forbes, M., & Choi, Y. (2020). The Curious Case of Neural Text Degeneration. *International Conference on Learning Representations*. https://openreview.net/forum?id=rygGQyrFvH

Kim, S., Hooper, C., Gholami, A., Dong, Z., Li, X., Shen, S., Mahoney, M. W., & Keutzer, K. (2024). SqueezeLLM: Dense-and-Sparse Quantization [_eprint: 2306.07629].

Kim, S., Mangalam, K., Moon, S., Malik, J., Mahoney, M. W., Gholami, A., & Keutzer, K. (2023). Speculative Decoding with Big Little Decoder. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt & S. Levine (Hrsg.), *Advances in Neural Information Processing Systems* (S. 39236–39256, Bd. 36). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2023/file/7b97adeafa1c51cf65263459ca9d0d7c-Paper-Conference.pdf

Least-Squares Method. (2008). In *The Concise Encyclopedia of Statistics* (S. 304–306). Springer New York. https://doi.org/10.1007/978-0-387-32833-1_228

Leviathan, Y., Kalman, M., & Matias, Y. (2023). Fast Inference from Transformers via Speculative Decoding [_eprint: 2211.17192]. https://arxiv.org/abs/2211.17192

LysandreJik. (2023, Mai). Release v4.29.0: Transformers Agents, SAM, RWKV, FocalNet, OpenLLaMa · huggingface/transformers. Verfügbar 28. April 2024 unter https://github.com/huggingface/transformers/releases/tag/v4.29.0

LysandreJik. (2024, April). Release v4.40.0: Llama 3, Idefics 2, Recurrent Gemma, Jamba, DBRX, OLMo, Qwen2MoE, Grounding Dino · huggingface/transformers. Verfügbar 28. April 2024 unter https://github.com/huggingface/transformers/releases/tag/v4.40.0

Patel, D., & Afzal, A. (2023, September). The Inference Cost Of Search Disruption – Large Language Model Cost Analysis. Verfügbar 26. April 2024 unter https://www.semianalysis.com/p/the-inference-cost-of-search-disruption

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2020). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*, *21*(140), 1–67. http://jmlr.org/papers/v21/20-074.html

Saxena, A. (2024, April). apoorvumang/prompt-lookup-decoding [original-date: 2023-11-24T13:11:47Z]. Verfügbar 28. April 2024 unter https://github.com/apoorvumang/prompt-lookup-decoding

Shazeer, N. (2019). Fast Transformer Decoding: One Write-Head is All You Need [_eprint: 1911.02150]. https://arxiv.org/abs/1911.02150

Shi, C., Yang, H., Cai, D., Zhang, Z., Wang, Y., Yang, Y., & Lam, W. (2024). A Thorough Examination of Decoding Methods in the Era of LLMs [_eprint: 2402.06925].

SleepyCal. (2015, Juni). Answer to "Dynamically mixin a base class to an instance in Python". Verfügbar 26. April 2024 unter https://stackoverflow.com/a/31075641

Stern, M., Shazeer, N., & Uszkoreit, J. (2018). Blockwise Parallel Decoding for Deep Autoregressive Models. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi & R. Garnett (Hrsg.), *Advances in Neural Information Processing Systems* (Bd. 31). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2018/file/c4127b9194fe8562c64dc0f5bf2c93bc-Paper.pdf

Vajapeyam, S. (2014). Understanding Shannon's Entropy metric for Information [_eprint: 1405.2061].

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., Platen, P. v., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., ... Rush, A. M. (2020). Transformers: State-of-the-Art Natural Language Processing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 38–45. https://www.aclweb.org/anthology/2020.emnlp-demos.6

Xiao, Y., & Watson, M. (2019). Guidance on Conducting a Systematic Literature Review [_eprint: https://doi.org/10.1177/0739456X17723971]. *Journal of Planning Education and Research*, *39*(1), 93–112. https://doi.org/10.1177/0739456X17723971

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., Mihaylov, T., Ott, M., Shleifer, S., Shuster, K., Simig, D., Koura, P. S., Sridhar, A., Wang, T., & Zettlemoyer, L. (2022). OPT: Open Pre-trained Transformer Language Models [_eprint: 2205.01068].

A. Appendix

A.1. Repository Entropy Drafting

Weitere Informationen zu Entropy Drafting und Source Code der Implementation finden sich unter: https://github.com/klfx/entropy-drafting

A.2. Hyperparameter Suche Token-Generation Messungen

GPU	m_t	m_d	Mean Target Time	Mean Draft Time	Ratio Target/Draft
A4000	OPT-2.7B	OPT-125M	34.15 ms	6.54 ms	5.22x
A4000	OPT-1.3B	OPT-125M	18.60 ms	6.54 ms	2.84x
A4000	OPT-350M	OPT-125M	11.70 ms	6.47 ms	1.81x
A4000	OPT-125M	OPT-125M	6.13 ms	6.55 ms	0.94x
A40	OPT-6.7B	OPT-125M	51.07 ms	8.32 ms	6.14x
A40	OPT-2.7B	OPT-125M	24.89 ms	8.20 ms	3.03x
A40	OPT-1.3B	OPT-125M	15.80 ms	8.42 ms	1.88x
A40	OPT-350M	OPT-125M	15.67 ms	8.35 ms	1.88x
A40	OPT-125M	OPT-125M	8.02 ms	8.23 ms	0.97x

Tabelle A.1.: Messdaten: Generierungskosten über verschiedene Modellgrössen

A.3. Ausführliche Ergebnisse OPT-6.7B

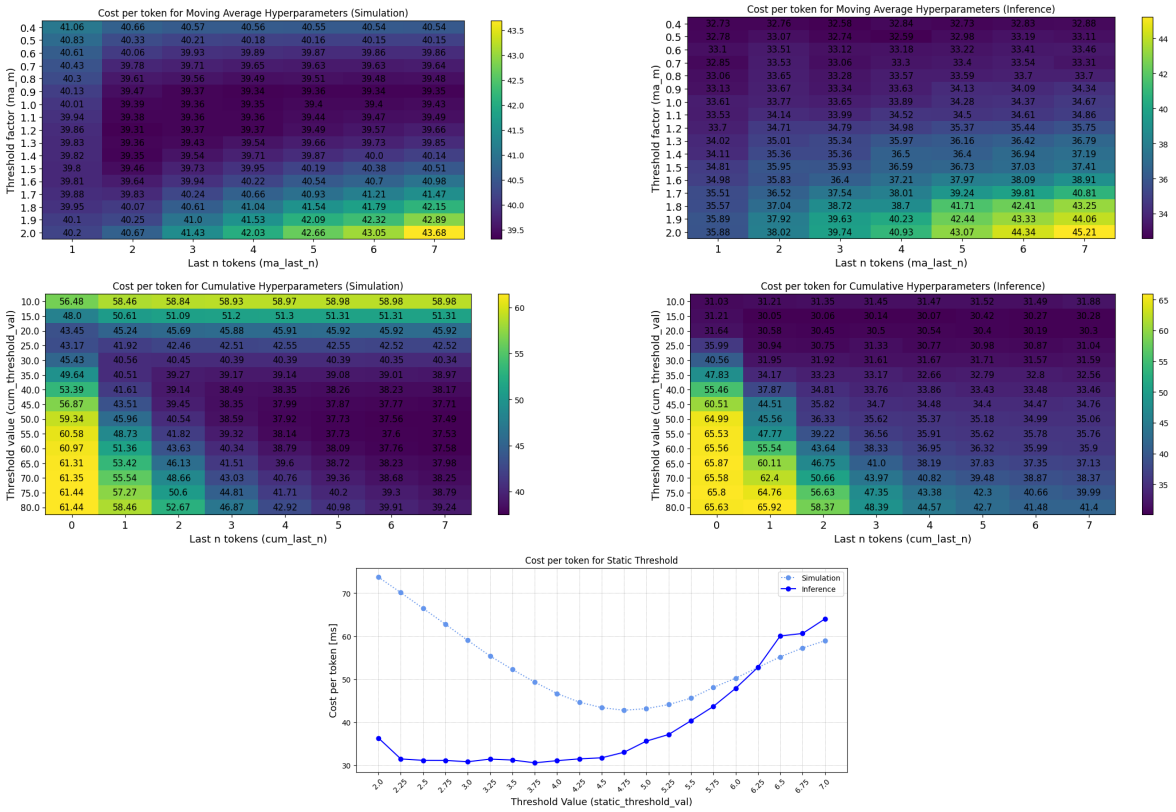


Abbildung A.1.: Hyperparameter Ergebnisse für Erhöhung der Modellgrösse (OPT-6.7B)

A.4. Ausführliche Ergebnisse Erhöhung Input-/Ausgabelänge

A.4.1. Experiment 1

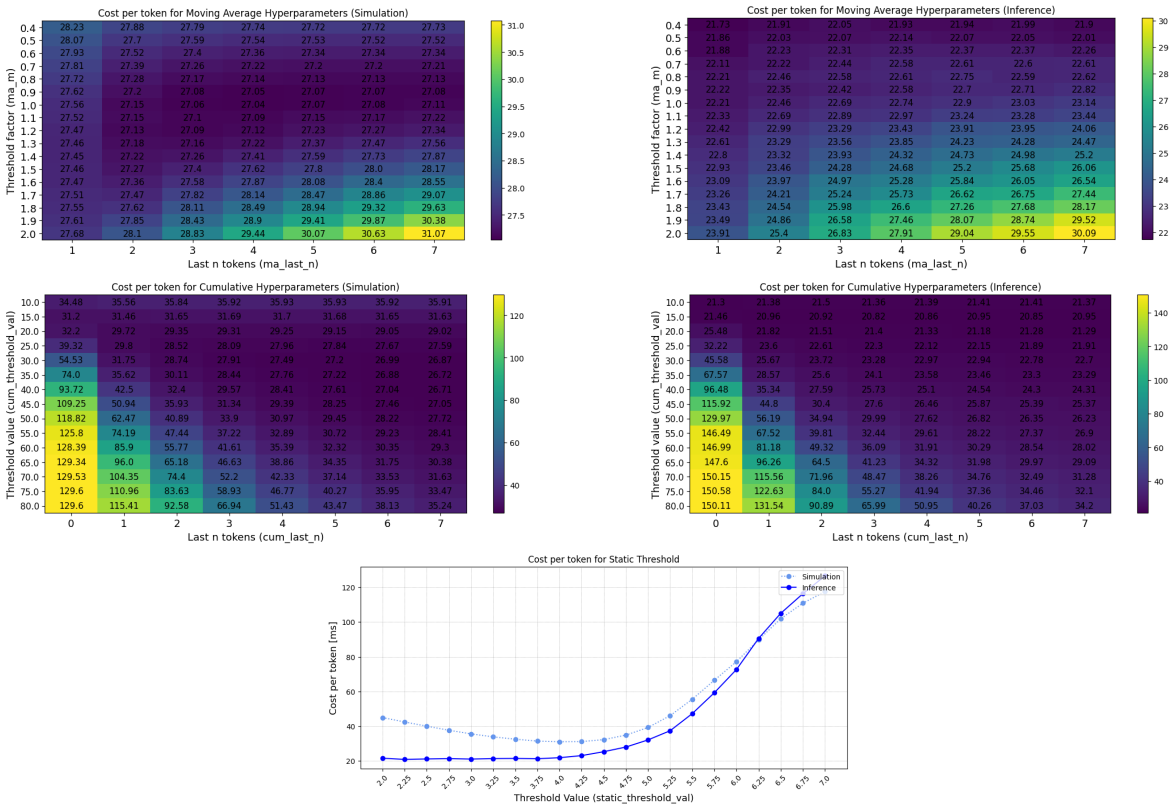


Abbildung A.2.: Hyperparameter Ergebnisse für Erhöhung der Länge (Experiment 1)

A.4.2. Experiment 2

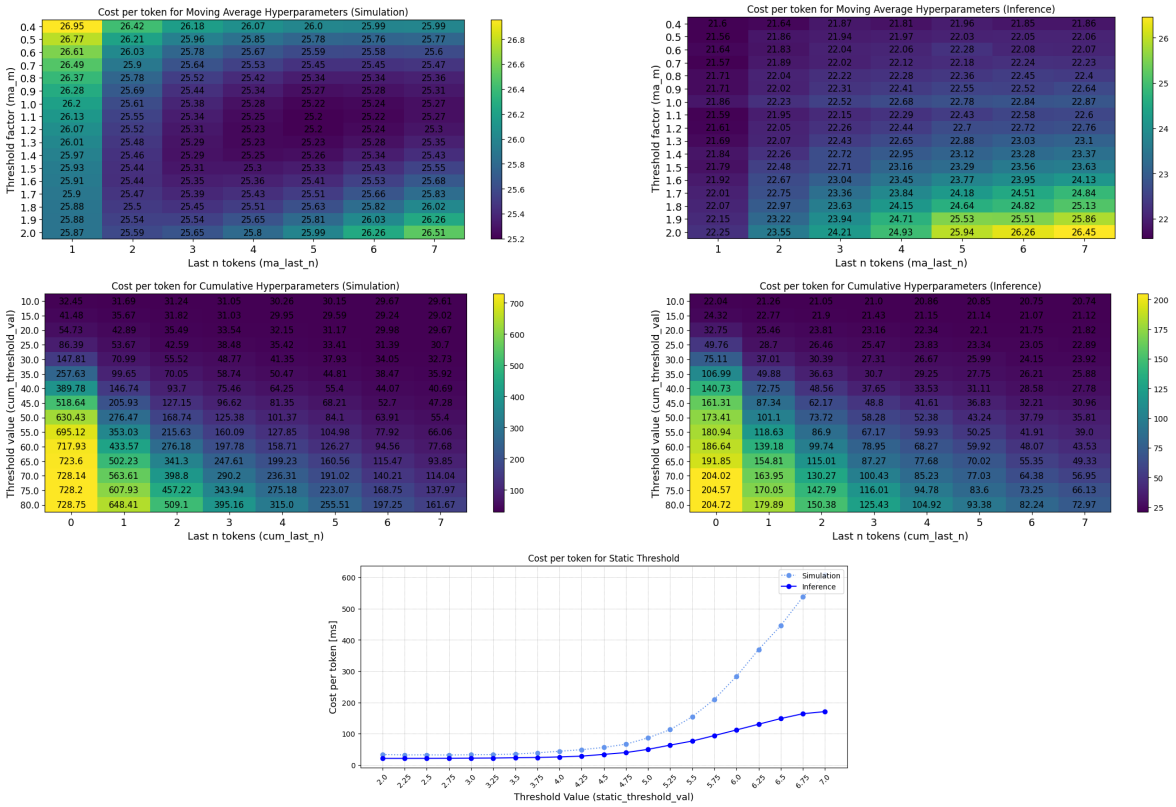


Abbildung A.3.: Hyperparameter Ergebnisse für Erhöhung der Länge (Experiment 2)

A.5. Aufgabenstellung

HSLU Hochschule Luzern Informatik

Aufgabenstellung

Modul:	Dept I BAA FS24

Titel:	Effiziente Text-Generierung mit spekulativem Decoding
Ausgangslage und Problemstellung:	Grosse Sprachmodelle (LLMs) beantworten Fragen, fassen Artikel zusammen und schreiben Texte in menschenähnlicher Qualität. Die Generierung von Text benötigt allerdings enorm viele Rechenressourcen. Im letzten Jahr wurden mehrere Techniken entwickelt, um diese Generierung effizienter zu machen. Eine davon ist Assisted Generation - auch Speculative Decoding genannt - bei der ein kleineres Modell den Text vorhersagt, den ein grösseres Modell der selben Modellfamilie wahrscheinlich generieren wird. Dadurch muss man das grössere Modell für wesentlich weniger Schritte laufen lassen, und kann den Rechenaufwand um das 2- bis 3-fache reduzieren. Mehr Details und eine Beschreibung der Technik: https://huggingface.co/blog/assisted-generation
Ziel der Arbeit und erwartete Resultate:	Nach einer Diskussion mit Joao Gante von Hugging Face bestehen mehrere Möglichkeiten, die Methode weiter zu verbessern. Das Ziel der Arbeit ist es, eine der folgenden Methoden auszuwählen, zu implementieren, und systematisch zu evaluieren. - Basierend auf der Entropie der Decoding-Verteilung von nächsten Wörtern kann vorhergesagt werden, wie sicher/unsicher sich das kleinere Modell bei der Dekodierung des nächsten Wortes ist. Anstelle einer fixen Zahl von weiteren Tokens, die dekodiert werden sollen (wie im oben genannten Artikel), kann eine adaptive Anzahl Tokens basierend auf der Decoding-Entropie generiert werden. - Durch das Generieren mehrere Kandidaten (z.B. mit Beam Search) kann die Übereinstimmung mit dem grösseren Modell verbessert und somit die Effizienz gesteigert werden.
Gewünschte Methoden, Vorgehen:	Basierend auf der Codebase von Hugging Face (https://github.com/gante/huggingface-demos/tree/main/experiments/faster_generation) sollen Experimente mit open-source LLMs (OPT, Pythia, LLaMA, Falcon) durchgeführt werden, die unterschiedliche Modellgrössen veröffentlicht haben. Zusätzliche Effizienz-Techniken wie Quantization werden nötig sein, um grosse Modelle auf einer einzelnen GPU laufen zu lassen. Eventuell können sehr grosse Modelle (Falcon-40B, LLaMA-70B) auch auf mehreren GPUs laufen gelassen werden, was eine zusätzliche Ingenieurs-Herausforderung bedeutet.
Kreativität, Methoden, Innovation:	- Studierende können eigene Ideen zur Verbesserung von Assisted Generation einbringen. - Als Ergänzung oder Variante dieser Arbeit kann das aktuell vorherrschende Prinzip von next token prediction auf ein Phrasen-basiertes Decoding erweitert werden. Phrasen sind Abfolgen von Tokens mit niedriger Entropie. - In der Zwischenzeit wurden weitere Ansätze veröffentlicht (e.g. MEDUSA, EAGLE), die Vorteile im Vergleich zu Speculative Decoding versprechen. Auf optionaler Basis können diese Ansätze oder Teile davon einbezogen werden.

Sonstige Bemerkungen:	Kandidat*innen müssen den NLP-Kurs im AI/ML-Bachelor bestanden haben.

Projektteam

Student:in 1:	Flavio Kluser
Betreuer:in:	Marfurt

Auftraggeber

Firma:	Hochschule Luzern
Ansprechperson:	Andreas Marfurt
Funktion:	Dozent
Strasse:	
PLZ/Ort:	
Telefon:	*<redacted>*
E-Mail:	andreas.marfurt@hslu.ch
Website:	

Version 13.06.2023 / bcl