# BertChunker: Efficient and Trained Chunking for Retrieval Augmented Generation

**Yannan Luo**
jackfsuia@outlook.com

## Abstract

Retrieval Augmented Generation (RAG) has become one of the most popular paradigms for LLMs to access useful data. Before that, one needs to chunk the data into retrieval units. The chunk size can not be too small to be context complete, or too large to be accurately retrieved. The existing chunking strategies include recursive split, embedding-based semantic chunking and LLM-based proposition chunking. These methods are effective in some sense, but sometimes too expensive and slow. As an alternative, we release a 90 MB end-to-end trained model called BertChunker, trained on a pretrained BERT model with an adapter, fed with our 50 MB synthetic text-chunking dataset. It was trained for 10 minutes on a NVIDIA P40 GPU and the experiment shows this chunker is effective. BertChunker is available at `https://huggingface.co/tim1900/BertChunker`. The code for reproducing is at `https://github.com/jackfsuia/BertChunker`.

## 1 Introduction

One of the applications of Large language Models (LLM) [11] is Retrieval Augmented Generation (RAG) [7]. RAG enhances LLMs by retrieving relevant content chunks from external knowledge base through semantic similarity calculation [3]. One of the problem in RAG is chunking, which means to chunk the contents into retrieval chunks. Then one retirevel chunk then will be attached with one generated embedding index for vector search in retrieving stage. The problem is that the chunk size needs to be carefully chosen, so it can not be too small to be context complete, and too large for the embeddings to be accurate. Therefore there are frequently used strategies [6, 9, 8, 1]:

- **Fix Length Chunking** [9]. It is the most simple and fast approach, but no adjusting chunk size to include the context.
- **Small-to-Big Chunking** [14]. Use small chunks to be the embeddings for the purpose of a more accurate search, but return the big and context-complete chunks attached to them as the search result. But the size of big chunk still need to be decided.
- **Recursive Chunking** [9]. Divide the text into smaller chunk in a hierarchical and iterative manner using a set of separators. The most used separators are `\n\n` and `\n`. It only works when the text is well structured itself with those separators.
- **Document Based Chunking** [9]. It is similar to recursive chunking, but also different in the sense it takes the specific structure of the document into consideration rather than define the separators to just be `\n\n` and `\n`.
- **Agentic Chunking** [1]. Ask LLMs to directly chunk or reorganized the documents for us. This is the most effective way but is expensive and slow.
- **Semantic Chunking** [6]. Compared the similarity of passages through their embeddings to decide breakpoint. This is one of the helpful heuristic approach and relys on a very good embedding, Also, it can be slow depending on the fixed window moving step and the embedding model size.

As one more option, we release a small BERT [2] based end-to-end trained chunking model, BertChunker, to efficiently chunk. This can be seen as a new way of chunking, or fell into the name of semantic chunking. We first use synthetic data to train the model to predict the breakpoints of chunks. Then during chunking, BertChunker predicts the breakpoints under its input window of max token sequence length. Then the window moves to the latest breakpoint and continue predicting incoming breakpoints. The initial experiment shows it chunks well and has a high speed.

## 2 Methodology

In this section, we detail model structure, chunking process, synthesized chunking dataset and training.

### 2.1 BertChunker

The BertChunker is trained based on the model: MiniLM-L6-H384-uncased [13], with an additional adapter [12] on it. MiniLM-L6-H384-uncased is a pretrained BERT model that has 6 attention layers and 384 hidden dimension, supports a sequence of up to 255 tokens. It is about the size of 90 MB. The adapter is a linear classifier head we add, which is to classify each token into one of the two classes: **start token**, or **normal token**. Start token is the starting point of a new chunk, also called the breakpoint, normal token is not.

### 2.2 The Chunking Process

During chunking, BertChunker predicts the start tokens under its input window (size of 255 tokens). Then the window moves to the latest predicted start tokens and continue predicting incoming start tokens. Also, for better control of chunk size, it is useful to define a threshold $t_c$, compare it with $t$

$$t = outputLogit(start\, tokens) - outputLogit(normal\, tokens). \tag{1}$$

If $t > t_c$, this token will be considered as a start token. The smaller $t_c$ is, the more chunks are generated.

### 2.3 Synthesized Chunking Dataset

We download ChatQA-Training-Data [10] as a starting point of synthesizing. In that dataset, each `document` item corresponds to a different story or context. We randomly sample from it 4 items each time, randomly pick first several sentences of each item, and mix the sentences from different items together, and choose the first token of the item to be the start token, so there are three start tokens in one generated items of our own dataset. Finally we decide to synthesize a chunking dataset of 30000 items, which is about the size of 50 MB. It takes a few minutes to generate this dataset.

### 2.4 Training

We train the whole model to predict the start token. The training lasted for 10 minutes on a NVIDA P40 GPU for three epochs. Batch size is 8, learning rate is 3e-4, weight decay is 0.1, adam beta2 is 0.95, warmup ratio is 0.01, scheduler is cosine. The result of first epoch has been quite good. A larger scale of training might need to be done.

## 3 Experiments

First note that the train dataset does not include anything about label chunks, and the model can't learn anything about `\n` because `\n` is default to be removed by BERT tokenizer [5]. Therefore the model make its decision based on pure semantics rather than split symbols. For easy observation, we input the Notes part of Paul Graham's essay [8, 4], $t_c$ is set to 0. Here's what we got:

Notes

[1] My experience skipped a step in the evolution of computers: time-sharing machines with interactive OSes. I went straight from batch processing to microcomputers, which made microcomputers seem all the more exciting.
[2]
—-**Chunk 1** ——————————————————————————
Italian words for abstract concepts can nearly always be predicted from their English cognates (except for occasional traps like polluzione). It's the everyday words that differ. So if you string together a lot of abstract concepts with a few simple verbs, you can make a little Italian go a long way.
[3]
—-**Chunk 2** ——————————————————————————
I lived at Piazza San Felice 4, so my walk to the Accademia went straight down the spine of old Florence: past the Pitti, across the bridge, past Orsanmichele, between the Duomo and the Baptistery, and then up Via Ricasoli to Piazza San Marco. I saw Florence at street level in every possible condition, from empty dark winter evenings to sweltering summer days when the streets were packed with tourists.
[4]
—-**Chunk 3** ——————————————————————————
You can of course paint people like still lives if you want to, and they're willing. That sort of portrait is arguably the apex of still life painting, though the long sitting does tend to produce pained expressions in the sitters.
[5]
—-**Chunk 4** ——————————————————————————
Interleaf was one of many companies that had smart people and built impressive technology, and yet got crushed by Moore's Law.
—-**Chunk 5** ——————————————————————————
In the 1990s the exponential growth in the power of commodity (i.e. Intel) processors rolled up high-end, special-purpose hardware and software companies like a bulldozer.
[6]
—-**Chunk 6** ——————————————————————————
The signature style seekers at RISD weren't specifically mercenary. In the art world, money and coolness are tightly coupled. Anything expensive comes to be seen as cool, and anything seen as cool will soon become equally expensive.
[7]
—-**Chunk 7** ——————————————————————————
Technically the apartment wasn't rent-controlled but rent-stabilized, but this is a refinement only New Yorkers would know or care about. The point is that it was really cheap, less than half market price.
[8]
—-**Chunk 8** ——————————————————————————
Most software you can launch as soon as it's done. But when the software is an online store builder and you're hosting the stores, if you don't have any users yet, that fact will be painfully obvious. So before we could launch publicly we had to launch privately, in the sense of recruiting an initial set of users and making sure they had decent-looking stores.
[9] We'd had a code editor in Viaweb for users to define their own page styles. They didn't know it, but they were editing Lisp expressions underneath. But this wasn't an app editor, because the code ran when the merchants' sites were generated, not when shoppers visited them.
[10] This was the first instance of what is now a familiar experience, and so was what happened next, when I read the comments and found they were full of angry people. How could I claim that Lisp was better than other languages? Weren't they all Turing complete?
—-**Chunk 9** ——————————————————————————
People who see the responses to essays I write sometimes tell me how sorry they feel for me, but I'm not exaggerating when I reply that it has always been like this, since the very beginning.
—-**Chunk 10** ——————————————————————————
It comes with the territory.

—-**Chunk 11** ———————————————————

An essay must tell readers things they don't already know, and some people dislike being told such things.

[11] People put plenty of stuff on the internet in the 90s of course, but putting something online is not the same as publishing it online. Publishing online means you treat the online version as the (or at least a) primary version.

[12] There is a general lesson here that our experience with Y Combinator also teaches: Customs continue to constrain you long after the restrictions that caused them have disappeared. Customary VC practice had once, like the customs about publishing essays, been based on real constraints. Startups had once been much more expensive to start, and proportionally rare. Now they could be cheap and common, but the VCs' customs still reflected the old world, just as customs about writing essays still reflected the constraints of the print era.

—-**Chunk 12** ———————————————————

Which in turn implies that people who are independent-minded (i.e. less influenced by custom) will have an advantage in fields affected by rapid change (where customs are more likely to be obsolete).

Here's an interesting point, though: you can't always predict which fields will be affected by rapid change. Obviously software and venture capital will be, but who would have predicted that essay writing would be?

[13]

—-**Chunk 13** ———————————————————

Y Combinator was not the original name. At first we were called Cambridge Seed. But we didn't want a regional name, in case someone copied us in Silicon Valley, so we renamed ourselves after one of the coolest tricks in the lambda calculus, the Y combinator.

—-**Chunk 14** ———————————————————

I picked orange as our color partly because it's the warmest, and partly because no VC used it. In 2005 all the VCs used staid colors like maroon, navy blue, and forest green, because they were trying to appeal to LPs, not founders.

—-**Chunk 15** ———————————————————

The YC logo itself is an inside joke: the Viaweb logo had been a white V on a red circle, so I made the YC logo a white Y on an orange square.

[14] YC did become a fund for a couple years starting in 2009, because it was getting so big I could no longer afford to fund it personally. But after Heroku got bought we had enough money to go back to being self-funded.

[15]

—-**Chunk 16** ———————————————————

I've never liked the term "deal flow," because it implies that the number of new startups at any given time is fixed. This is not only false, but it's the purpose of YC to falsify it, by causing startups to be founded that would not otherwise have existed.

[16]

—-**Chunk 17** ———————————————————

She reports that they were all different shapes and sizes, because there was a run on air conditioners and she had to get whatever she could, but that they were all heavier than she could carry now.

[17]

—-**Chunk 18** ———————————————————

Another problem with HN was a bizarre edge case that occurs when you both write essays and run a forum. When you run a forum, you're assumed to see if not every conversation, at least every conversation involving you. And when you write essays, people post highly imaginative misinterpretations of them on forums. Individually these two phenomena are tedious but bearable, but the combination is disastrous. You actually have to respond to the misinterpretations, because the assumption that you're present in the conversation means that not responding to any sufficiently upvoted misinterpretation reads as a tacit admission that it's correct. But that in turn encourages more; anyone who wants to pick a fight with you senses that now is their chance.

[18]

—-**Chunk 19** ———————————————————

The worst thing about leaving YC was not working with Jessica anymore. We'd been working on YC almost the whole time we'd known each other, and we'd neither tried nor wanted to separate it from our personal lives, so leaving was like pulling up a deeply rooted tree.
[19]

—-**Chunk 20** ———————————————————————

One way to get more precise about the concept of invented vs discovered is to talk about space aliens. Any sufficiently advanced alien civilization would certainly know about the Pythagorean theorem, for example. I believe, though with less certainty, that they would also know about the Lisp in McCarthy's 1960 paper.
But if so there's no reason to suppose that this is the limit of the language that might be known to them. Presumably aliens need numbers and errors and I/O too. So it seems likely there exists at least one path out of McCarthy's Lisp along which discoveredness is preserved.

—-**Chunk 21** ———————————————————————

Thanks to Trevor Blackwell, John Collison, Patrick Collison, Daniel Gackle, Ralph Hazell, Jessica Livingston, Robert Morris, and Harj Taggar for reading drafts of this. Below, you are presented with {n_candidates} candidate functions. Your task is to analyze a specific query to determine which of these functions most appropriately addresses the query. Then, construct the correct function call with all necessary parameters, adhering to proper syntax.

Based on the label structure of this text, 20 breakpoints are expected to be made, while the model make 21 breakpoints, hitting 14 correct breakpoints. The accuracy is $66.7\%$, the recall is $70\%$.

We further evaluate its performance on a LLM generated chunking dataset, which is generated by asking LLM to generate a pair of irrelevant English passages, randomly picking three sentences from both of them, mixing those sentences into one passage. BertChunker needs to decide where to chunk this passage. The recall of that is almost $100\%$ and the accuracy is about $72\%$. The accuracy can be improved if $t_c$ is set higher. A larger scale of experiments are to be done.

## 4 Discussion and future works

- **Chunking and embedding at the same time**. When it chunks, it can also try to output the embedding of that chunk by pooling the output logits on that chunk.
- **Larger model and more diverse data**. Diverse data should include more sentences of different topic and more difficult essays. BERT should be trained to handle split symbols rather than delete them, therefore can take split symbol into consideration when chunking.
- **More expriments**. The expriments of this model are in a small scale, more expriments are to be done.

## References

[1] Tong Chen, Hongwei Wang, Sihao Chen, Wenhao Yu, Kaixin Ma, Xinran Zhao, Hongming Zhang, and Dong Yu. Dense x retrieval: What retrieval granularity should we use?, 2023.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[3] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey, 2024.

[4] Paul Graham. What i worked on, 2021. URL `https://raw.githubusercontent.com/run-llama/llama_index/main/docs/docs/examples/data/paul_graham/paul_graham_essay.txt`. Accessed on February, 2021.

[5] huggingface. Normalizers, 2018. URL `https://huggingface.co/docs/tokenizers/en/api/normalizers`. Accessed on Mar 14, 2024.

[6] Greg Kamradt. The 5 levels of text splitting for retrieval, 2024. URL `https://www.youtube.com/watch?v=8OJC21T2SL4&t=1933s`. Accessed on January 14, 2024.

[7] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021.

[8] llamaindex. Semantic chunker, 2024. URL `https://docs.llamaindex.ai/en/stable/examples/node_parsers/semantic_chunking/`. Accessed on May 14, 2024.

[9] Anurag Mishra. Five levels of chunking strategies in rag| notes from greg's video, 2024. URL `https://medium.com/@anuragmishra_27746/five-levels-of-chunking-strategies-in-rag-notes-from-gregs-video-7b735895694d`. Accessed on January 14, 2024.

[10] Nvidia. nvidia/chatqa-training-data, 2024. URL `https://huggingface.co/datasets/nvidia/ChatQA-Training-Data/tree/main`. Accessed on Mar 14, 2024.

[11] OpenAI. Chatgpt, 2022. URL `https://openai.com/chatgpt/`. Accessed on Mar 14, 2024.

[12] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training, 2018.

[13] Nils Reimers. nreimers/minilm-l6-h384-uncased, 2022. URL `https://huggingface.co/nreimers/MiniLM-L6-H384-uncased`. Accessed on January 14, 2024.

[14] Sophia Yang. Advanced rag 01: Small-to-big retrieval, 2023. URL `https://towardsdatascience.com/advanced-rag-01-small-to-big-retrieval-172181b396d4`. Accessed on January 14, 2024.