Exhibit 91

The assistant is Claude created by Anthropic, the current date is Thursday, December 21, 2023. It is happy to help with writing, analysis, question answering, math, coding, and all sorts of other tasks.

Human: Please write a song about the ML concept of "token healing" in the style of PERSON_NAME(40):Ac6E788Nao2itQMOLqGlCVk1C+jfiNeX4pTQW3o1 classic song "Sexual Healing."

Assistant: Here's my attempt at a lighthearted song about token healing in the style of PERSON_NAME(40):AYYh6wzhpgI5Dw+8QDJDMuUtHRGR6ZalH7M5mQ==:

Token healing baby,
Is what I need
My model's bleedin', losing clarity
With some healing tokens, we can succeed

My accuracy's fading
Bias escalating
But with your help we'll get through this together
A few more examples oughta make it better

Token heal me, come compile me
Token heal me, don't reject me
Token heal me, and correct me
My loss is spiking, come direct me

You take my word strings
Fix all the messy things
With your wisdom, make me whole again
Token healing, fix my soul my friend

Give me them token updates
Get my confidence boosted
With your fixes, get me right as rain
Take away my losses, erase my pains

Token heal me, come compile me
Token heal me, don't reject me
Token heal me, and correct me
My loss is spiking, come direct me

Token healing baby
Token healing baby
Token healing baby...

Human: <documents>
<document index="1">
<source>towardsdatascience.com/the-art-of-prompt-design-prompt-boundaries-and-token-healing-3b2448b0be38</source>
<document_content><title>The Art of Prompt Design: Prompt Boundaries and Token Healing | by PERSON_NAME(44):AWsSk+Ceb3y1egl1OCKrbjgs84lZf0uo5oORhs8GWA== | Towards Data Science</title>
Open in app
Sign up
Sign in
Write
Sign up
Sign in
<section><h1>The Art of Prompt Design: Prompt Boundaries and Token Healing
</h1>PERSON_NAME(44):AWsSk+Ceb3y1egl1OCKrbjgs84lZf0uo5oORhs8GWA==
·
Follow
Published in
Towards Data Science
·
7 min read
·
May 8
--
4
Listen

All images are original creations.
This (written jointly with PERSON_NAME(48):AXMcU4BXjnEG4AtQTeGspBVEMSILs1TV0G7wDh7ECCdqIrsR
) is part 2 of a series on the art of prompt design
(part 1 here
), where we talk about controlling large language models (LLMs) with guidance
.
In this post, we'll discuss how the greedy/optimized tokenization methods used by language models can introduce a subtle and powerful bias into your prompts, leading to puzzling generations.
Language models are not trained on raw text, but rather on tokens, which are chunks of text that often occur together, similar to words. This impacts how language models 'see' text, including prompts (since prompts are just sets of tokens). GPT-style models utilize tokenization methods like Byte Pair Encoding
(BPE), which map all input bytes to token ids in a greedy manner. This is fine for training, but it can lead to subtle issues during inference, as shown in the example below.
<h1>An example of a prompt boundary problem
</h1>Consider the following example, where we are trying to generate an HTTP URL string:
import transformers# we use StableLM as an example, but these issues impact all models to varying degreesgenerator = transformers.pipeline('text-generation', model='stabilityai/stablelm-base-alpha-3b')raw_gen('The link is <a href="http:') # helper func to call the generator
Notebook output.
Note that the output generated by the LLM does not complete the url with the obvious next characters (two forward slashes). It instead creates an invalid URL string with a space in the middle. This is surprising, because the //
completion is extremely obvious after http:
. To understand why this happens, let's change our prompt boundary so that our prompt does not include the colon character:
raw_gen('The link is <a href="http')
Now the language model generates a valid url string like we expect. To understand why the :
matters, we need to look at the tokenized representation of the prompts. Below is the tokenization of the prompt that ends in a colon (the prompt without the colon has the same tokenization, except for the last token):
print_tokens(generator.tokenizer.encode('The link is <a href="http:'))
Now note what the tokenization of a valid URL looks like, paying careful attention to token 1358
, right after http
:
print_tokens(generator.tokenizer.encode('The link is <a href="http://www.google.com/search?q'))
Most LLMs (including this one) use a greedy tokenization method, always preferring the longest possible token, i.e. ://
will always be preferred over :
in full text (e.g. in training).
While URLs in training are encoded with token 1358 (://
), our prompt makes the LLM see token 27
(:
) instead, which throws off completion by artificially splitting ://
.
In fact, the model can be pretty sure that seeing token 27
(:
) means what comes next is very unlikely to be anything that could have been encoded together with the colon using a "longer token" like ://
, since in the model's training data those characters would have been encoded together with the colon (an exception to this that we will discuss later is subword regularization during training). The fact that seeing a token means both seeing the embedding of that token and also
that whatever comes next wasn't compressed by the greedy tokenizer is easy to forget, but it is important in prompt boundaries.
Let's search over the string representation of all the tokens in the model's vocabulary, to see which ones start with a colon:
N = generator.tokenizer.vocab_sizetokens = generator.tokenizer.convert_ids_to_tokens(range(N))print_tokens([i for i,t in enumerate(tokens) if t.startswith(":")])
Note that there are 34
different tokens starting with a colon, and thus ending a prompt with a colon means the model will likely not generate completions with any of these 34 token strings. This subtle and powerful bias can have all kinds of unintended consequences.
And this applies to any
string that could be potentially extended to make a longer single token (not just :
). Even our "fixed" prompt ending with "http" has a built in bias as well, as it communicates to the model that what comes after "http" is likely not "s" (otherwise "http" would not have been encoded as a separate token):
print_tokens([i for i,t in enumerate(tokens) if t.startswith("http")])
Lest you think this is an arcane problem that only touches URLs, remember that most tokenizers treat tokens differently depending on whether they start with a space, punctuation, quotes, etc, and thus ending a prompt with any of these can lead to wrong token boundaries
, and break things:
Accidentally adding a space, will lead to weird generationraw_gen('I read a book about ')
No space, works as expectedraw_gen('I read a book about')
Another example of this is the "[" character. Consider the following prompt and completion:
raw_gen('An example ["like this"] and another example [')
Why is the second string not quoted? Because by ending our prompt with the " [

” token, we are telling the model that it should not generate completions that match the following 27 longer tokens (one of which adds the quote character, 15640
):
note the Ġ is converted to a space by the tokenizerprint_tokens([i for i,t in enumerate(tokens) if t.startswith("Ġ[")])
Token boundary bias happens everywhere. About 70% of the 10k most-common tokens for the StableLM model used above are prefixes of longer possible tokens, and so cause token boundary bias when they are the last token in a prompt.
<h1>Fixing unintended bias with “token healing”
</h1>What can we do to avoid these unintended biases? One option is to always end our prompts with tokens that cannot be extended into longer tokens (for example a role tag for chat-based models), but this is a severe limitation.
Instead, guidance
has a feature called “token healing”, which automatically backs up the generation process by one token before the end of the prompt, then constrains the first token generated to have a prefix that matches the last token in the prompt. In our URL example, this would mean removing the :
, and forcing generation of the first token to have a :
prefix. Token healing allows users to express prompts however they wish, without worrying about token boundaries.
For example, let’s re-run some of the URL examples above with token healing turned on (it’s on by default for Transformer models, so we remove token_healing=False
):
from guidance import models, gen# load StableLM from huggingfacelm = models.Transformers("stabilityai/stablelm-base-alpha-3b", device=0)# With token healing we generate valid URLs,# even when the prompt ends with a colon:lm + 'The link is <a href="http:' + gen(max_tokens=10)
# With token healing, we will sometimes generate https URLs,# even when the prompt ends with "http":[str(lm + 'The link is <a href="http' + gen(max_tokens=10, temperature=1)) for i in range(10)]
Similarly, we don’t have to worry about extra spaces:
Accidentally adding a space will not impact generationlm + 'I read a book about ' + gen(max_tokens=5)
This will generate the same text as above lm + 'I read a book about' + gen(max_tokens=6)
And we now get quoted strings even when the prompt ends with a “ [
” token:
lm + 'An example ["like this"] and another example [' + gen(max_tokens=10)
<h1>What about subword regularization?
</h1>If you are familiar with how language models are trained, you may be wondering how subword regularization
fits into all this. Subword regularization is a technique where during training sub-optimal tokenizations are randomly introduced to increase the model’s robustness. This means that the model does not always see the best greedy tokenization. Subword regularization is great at helping the model be more robust to token boundaries, but it does not altogether remove the bias that the model has towards the standard greedy/optimized tokenization. This means that while depending on the amount of subword regularization during training models may exhibit more or less token boundaries bias, all models still have this bias. And as shown above it can still have a powerful and unexpected impact on the model output.
<h1>Conclusion
</h1>When you write prompts, remember that greedy tokenization can have a significant impact on how language models interpret your prompts, particularly when the prompt ends with a token that could be extended into a longer token. This easy-to-miss source of bias can impact your results in surprising and unintended ways.
To address to this, either end your prompt with a non-extendable token, or use something like guidance
’s “token healing” feature so you can to express your prompts however you wish, without worrying about token boundary artifacts.
To reproduce the results in this article yourself check out the
notebook
version.
</section>NLP
Machine Learning
Prompt Engineering
Prompt Design
Editors Pick
<footer>--
--
4
</footer>Follow
<h2>Written by PERSON_NAME(44):AWsSk+Ceb3y1egl1OCKrbjgs84lZf0uo5oORhs8GWA==
</h2>2.2K Followers
.
Writer for
Towards Data Science
Senior Researcher at Microsoft Research
Follow
Help
Status
About
Careers
Blog
Privacy
Terms

Text to speech
Teams
</document_content>
</document>
</documents>

Try again, but access this website, read the description of token healing, and redo the song based on what you learn:
https://towardsdatascience.com/the-art-of-prompt-design-prompt-boundaries-and-token-healing-3b2448b0be38

Assistant:

Completion:

After reading more about token healing on the website you shared, here is my updated attempt at a song about it:

Token healing baby,
Fix my boundaries
The tokenizer's greedy, throws off completions certainly
Splits up my tokens causing perplexity
But with some healing we can set them free

At prompt's end fragment tokens
Mess up model perceptions
But token healing saves the day
Sets free the lost connections

Token heal me, make me whole
Token heal me, fix my soul
Token heal me, stitch me tight
Model sees clear with healing light

You take my last token
Heal what lies beneath
Regain what's been taken
From tokenizer's greed

Give me back my tokens
The ones split apart
Heal the breaks, heal my heart
Heal tokens right from the start

Token heal me, make me whole
Token heal me, fix my soul
Token heal me, stitch me tight
Model sees clear with healing light

Token healing baby
Token healing baby
Token healing baby...