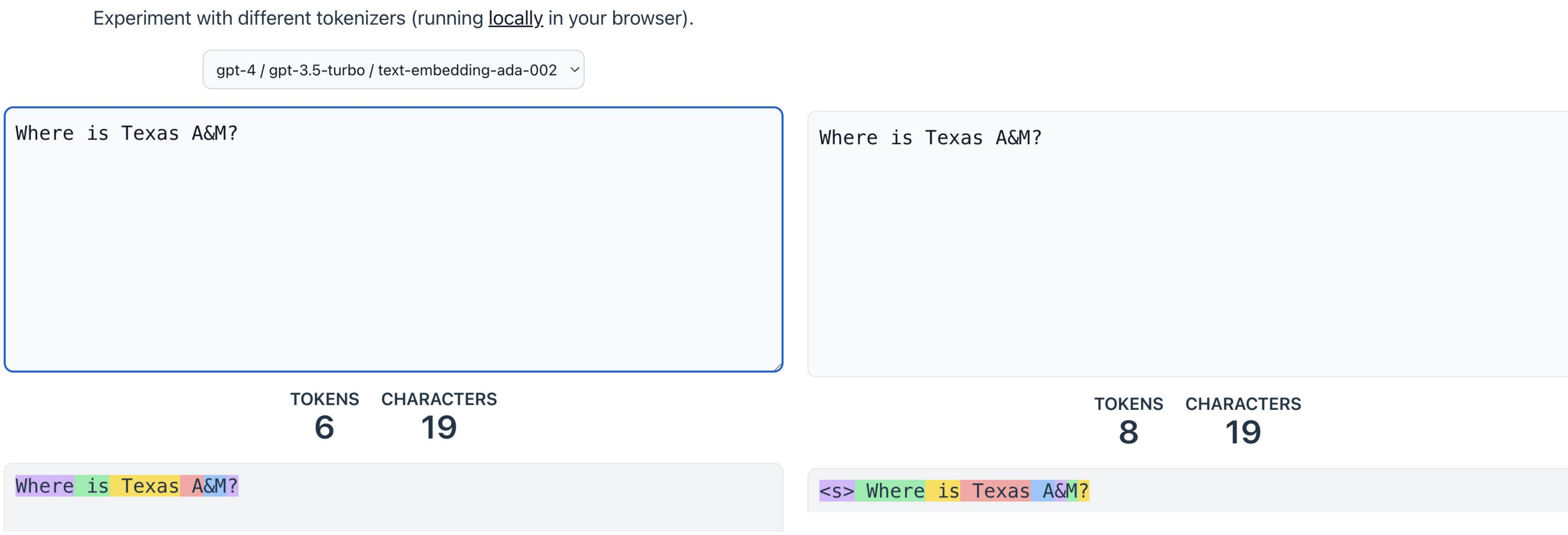
CSCE 689-609
Tokenization and Sampling

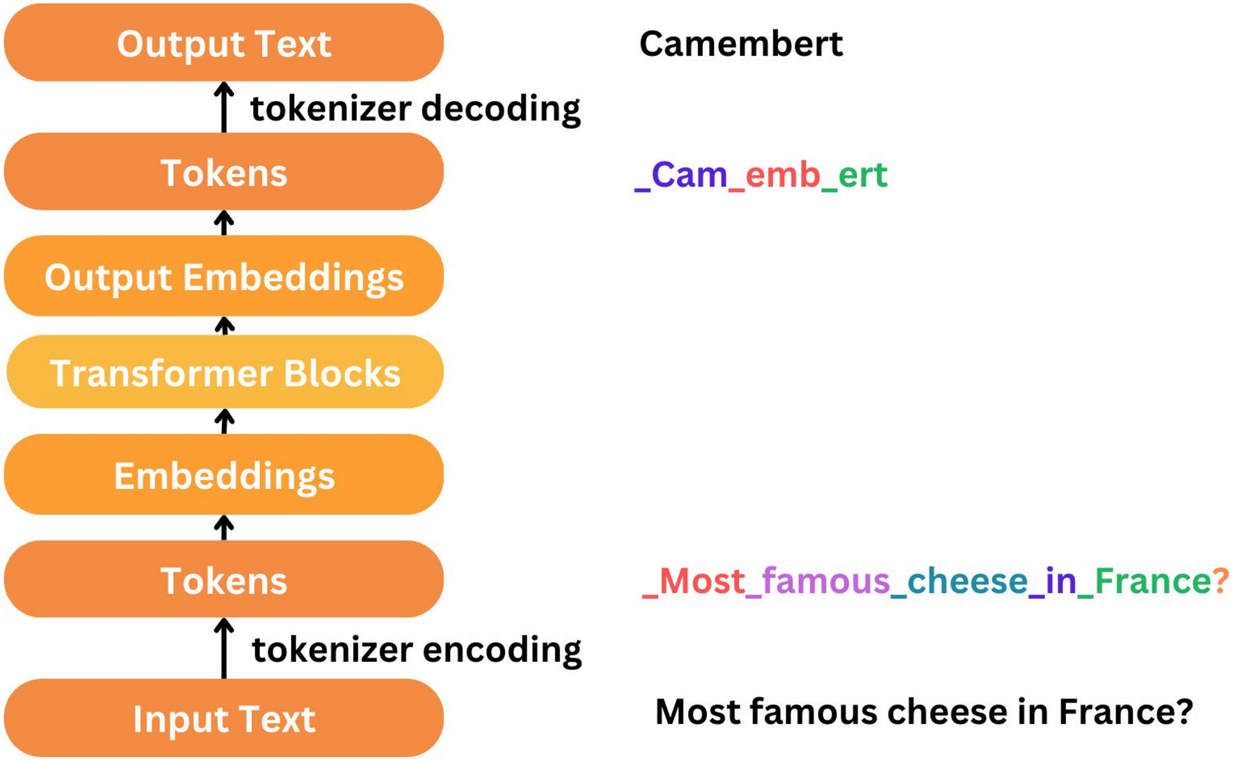
Jeff Huang
jeff@cse.tamu.edu
o2lab.github.io

The Tokenizer Playground

https://huggingface.co/spaces/Xenova/the-tokenizer-playground



Tokenization (e.g., **Mistral AI**: https://docs.mistral.ai/guides/tokenization)



Establishing the Vocabulary

1. **Collect Training Data**: Gather a large corpus of text data that the model will learn from.
2. **Initial Tokenization**: Apply preliminary tokenization methods to split the text into basic units (words, subwords, or characters).
3. **Vocabulary Creation**: Choose a tokenization algorithm (e.g., Byte Pair Encoding (BPE), WordPiece, Unigram, SentencePiece) to generate a manageable and efficient set of tokens.
4. **Apply Algorithm:** Run the selected algorithm on the initial tokens to create a set of subword tokens or characters that capture the linguistic nuances of the training data.
5. **Assign IDs:** Each unique token in the resulting vocabulary is assigned a specific integer ID.

Real-Time Tokenization Process

1. **Convert text stream to tokens**: Convert incoming text into the tokens found in the established vocabulary, ensuring all text can be represented.
2. **Token ID**: Map each token to its corresponding integer ID as defined in the pre-established vocabulary).

Implementation https://github.com/karpathy/llama2.c/blob/master/run.c#L452-L571

```c
void encode(Tokenizer* t, char *text, int8_t bos, int8_t eos, int *tokens, int *n_tokens) {
    // encode the string text (input) into an upper-bound preallocated tokens[] array
    // bos != 0 means prepend the BOS token (=1), eos != 0 means append the EOS token (=2)
    if (text == NULL) { fprintf(stderr, "cannot encode NULL text\n"); exit(EXIT_FAILURE); }

    if (t->sorted_vocab == NULL) {
        // lazily malloc and sort the vocabulary
        t->sorted_vocab = malloc(t->vocab_size * sizeof(TokenIndex));
        for (int i = 0; i < t->vocab_size; i++) {
            t->sorted_vocab[i].str = t->vocab[i];
            t->sorted_vocab[i].id = i;
        }
        qsort(t->sorted_vocab, t->vocab_size, sizeof(TokenIndex), compare_tokens);
    }

    // create a temporary buffer that will store merge candidates of always two consecutive tokens
    // *2 for concat, +1 for null terminator +2 for UTF8 (in case max_token_length is 1)
    char* str_buffer = malloc((t->max_token_length*2 +1 +2) * sizeof(char));
    size_t str_len = 0;
```

Tokenization Basic Units

- Word-level
- Character-level
- Subword-level
- Byte-level
- Multi-word
- Phrase-level
- ...

Token size	Pros	Cons
Smaller tokens (character or subword tokenization)	- Enables the model to handle a wider range of inputs, such as unknown words, typos, or complex syntax. - Might allow the vocabulary size to be reduced, requiring fewer memory resources.	- A given text is broken into more tokens, requiring additional computational resources while processing - Given a fixed token limit, the maximum size of the model's input and output is smaller
Larger tokens (word tokenization)	- A given text is broken into fewer tokens, requiring fewer computational resources while processing. - Given the same token limit, the maximum size of the model's input and output is larger.	- Might cause an increased vocabulary size, requiring more memory resources. - Can limit the models ability to handle unknown words, typos, or complex syntax.

Mistral V3 (tekken) tokenizer

- **Subword-level tokenization**
- **Byte-Pair Encoding (BPE)**
- **Vocabulary size**: 130k vocab + 1k control tokens
- **Special control tokens**: 14
    - The tokenizer does not encode control tokens (to prevent prompt injection)

```
<unk>
<s>
</s>
[INST]
[/INST]
[AVAILABLE_TOOLS]
[/AVAILABLE_TOOLS]
[TOOL_RESULTS]
[/TOOL_RESULTS]
[TOOL_CALLS]
<pad>
[PREFIX]
[MIDDLE]
[SUFFIX]
```

Mistral control tokens

AI21's Jurassic models tokenizers

https://docs.ai21.com/docs/large-language-models

Token dictionary

AI21 Studio uses a large token dictionary (250K), which contains tokens generated from separate characters, words, word parts, such as prefixes and suffixes, and multi-word tokens. For example, in our current tokenizer, the phrase "I want to break free." is split into the following tokens:

```
['_I_want_to', '_break', '_free', '.']
```

Byte-Pair Encoding (BPE)

Paper: Neural Machine Translation of Rare Words with Subword Units

- **Byte split:** It starts by treating each byte in a text as a separate token
- **Merge:** Then, it iteratively adds new tokens to the vocabulary for the most frequent pair of tokens currently appearing in the corpus.
  - E.g., if the most frequent pair of tokens is "th" + "e", then a new token "the" will be added.
- This process continues until a target vocabulary size is reached

Example: https://huggingface.co/docs/transformers/en/tokenizer_summary

- **Popular:** GPT-4, Claude, Llama-3, all use BPE

Unigram

Paper: [Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates]

- **Idea:** In contrast to BPE, Unigram initializes its base vocabulary to a large number of symbols and progressively trims down each symbol to obtain a smaller vocabulary.
- **Loss-based removal:** The algorithm defines a loss over the training data.
  - For each symbol, it computes how much the overall loss would increase if the symbol was to be removed from the vocabulary
  - It then removes the symbols whose loss increase is the lowest

SentencePiece

Paper: https://arxiv.org/pdf/1808.06226

- **Idea:** It treats the input text as a continuous sequence of characters without explicitly splitting it by whitespace (it escapes the whitespace with a meta symbol _ (U+2581))
    - For tokenization, it can use BPE or Unigram

- Examples of models using SentencePiece are ALBERT, XLNet, Marian, and T5.

Sampling (the Next Token)

Logits: Unnormalized scores that represent the likelihood of each token being the next in the sequence

- Logits are passed through a softmax function to convert into a probability distribution (sum to 1)

Temperature: Before sampling, the distribution can be adjusted using a parameter called **temperature (T):**

Probabilities

Tokens	Logits	Exponents	Softmax
	-109.13	1.00e+0	56.98%
The	-111.36	6.20e-2	3.53%
I	-111.46	5.48e-2	3.12%
It	-111.49	5.29e-2	3.02%
What	-111.53	4.98e-2	2.84%

$$p_i = \frac{\exp\left(\frac{\text{logit}_i}{T}\right)}{\sum_j \exp\left(\frac{\text{logit}_j}{T}\right)}$$

Top-k Sampling: only the top **k** most probable tokens are considered for sampling.

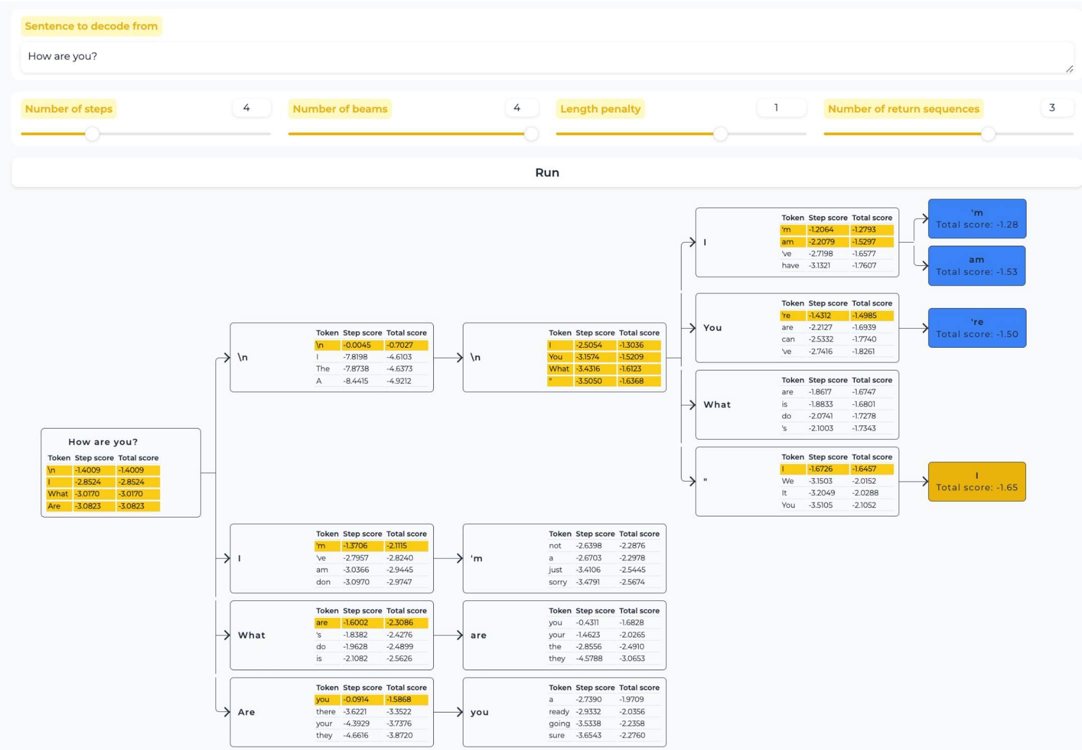
Top-p Sampling: chooses tokens from the smallest possible set whose cumulative probability exceeds a threshold **p**.

Implementation https://github.com/karpathy/llama2.c/blob/master/run.c#L691-L714

```c
int sample(Sampler* sampler, float* logits) {
    // sample the token given the logits and some hyperparameters
    int next;
    if (sampler->temperature == 0.0f) {
        // greedy argmax sampling: take the token with the highest probability
        next = sample_argmax(logits, sampler->vocab_size);
    } else {
        // apply the temperature to the logits
        for (int q=0; q<sampler->vocab_size; q++) { logits[q] /= sampler->temperature; }
        // apply softmax to the logits to get the probabilities for next token
        softmax(logits, sampler->vocab_size);
        // flip a (float) coin (this is our source of entropy for sampling)
        float coin = random_f32(&sampler->rng_state);
        // we sample from this distribution to get the next token
        if (sampler->topp <= 0 || sampler->topp >= 1) {
            // simply sample from the predicted probability distribution
            next = sample_mult(logits, sampler->vocab_size, coin);
        } else {
            // top-p (nucleus) sampling, clamping the least likely tokens to zero
            next = sample_topp(logits, sampler->vocab_size, sampler->topp, sampler->probindex, coin);
        }
    }
    return next;
}
```

Parallel Decoding (decode multiple parallel sequences)

https://huggingface.co/spaces/m-ric/beam_search_visualizer



https://platform.openai.com/docs/api-reference/chat/create

n

integer or null

Optional

Defaults to 1

How many chat completion choices to generate for each input message. Note that you will be charged based on the number of generated tokens across all of the choices. Keep n as 1 to minimize costs.

```
./llama-batched -m Meta-Llama-3.1-8B-Instruct-Q4_K_M.gguf -p "9.11 and 9.9, which one is larger? answer:" -np 4
```

Important Notes

- Read:
  - llama2.c: https://github.com/karpathy/llama2.c
  - SGLang https://github.com/sgl-project/sglang
- Due
  - HW1 (Saturday Sep 14, 11:59 PM)