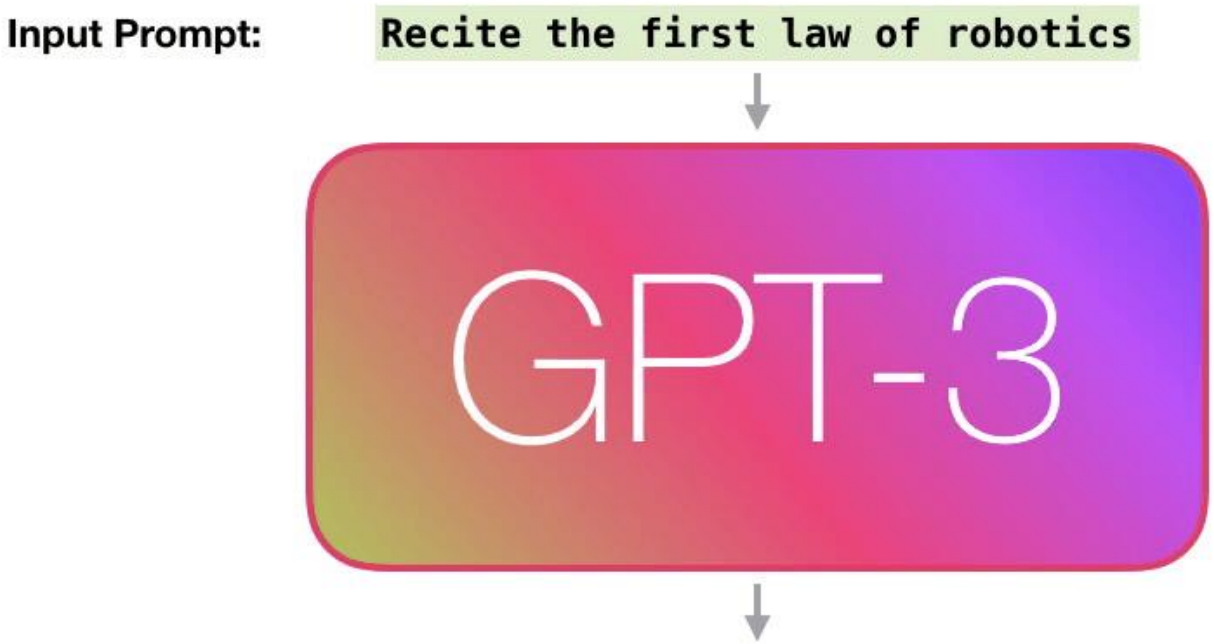
文本解码原理

以MindNLP为例

[M]S 昇思 MindSpore

GPT3是怎么工作的



自回归语言模型：根据前文预测下一个单词

Text: Second Law of Robotics: A robot must obey the orders given it by human beings

Generated training examples

Example #	Input (features)	Correct output (labels)
1	Second law of robotics :	a
2	Second law of robotics : a	robot
3	Second law of robotics : a robot	must
...		

自回归语言模型

一个文本序列的概率分布可以分解为每个词基于其上文的条件概率的乘积

$$P(w_{1:T}|W_0) = \prod_{t=1}^{T} P(w_t|w_{1:t-1}, W_0) \text{ ,with } w_{1:0} = \emptyset,$$

W_0:初始上下文单词序列

T: 时间步

当生成EOS标签时，停止生成。

MindNLP/huggingface Transformers 提供的文本生成方法

```python
class GenerationMode(ExplicitEnum):
    """
    Possible generation modes, downstream of the [`~generation.GenerationMixin.generate`] method.
    """

    # Non-beam methods
    CONTRASTIVE_SEARCH = "contrastive_search"
    GREEDY_SEARCH = "greedy_search"
    SAMPLE = "sample"
    ASSISTED_GENERATION = "assisted_generation"
    # Beam methods
    BEAM_SEARCH = "beam_search"
    BEAM_SAMPLE = "beam_sample"
    CONSTRAINED_BEAM_SEARCH = "constrained_beam_search"
    GROUP_BEAM_SEARCH = "group_beam_search"
```

Contents

- Greedy search
- Beam search
- Sample

- Constrained Beam search
- Contrastive Search
- Assisted Search

Greedy search

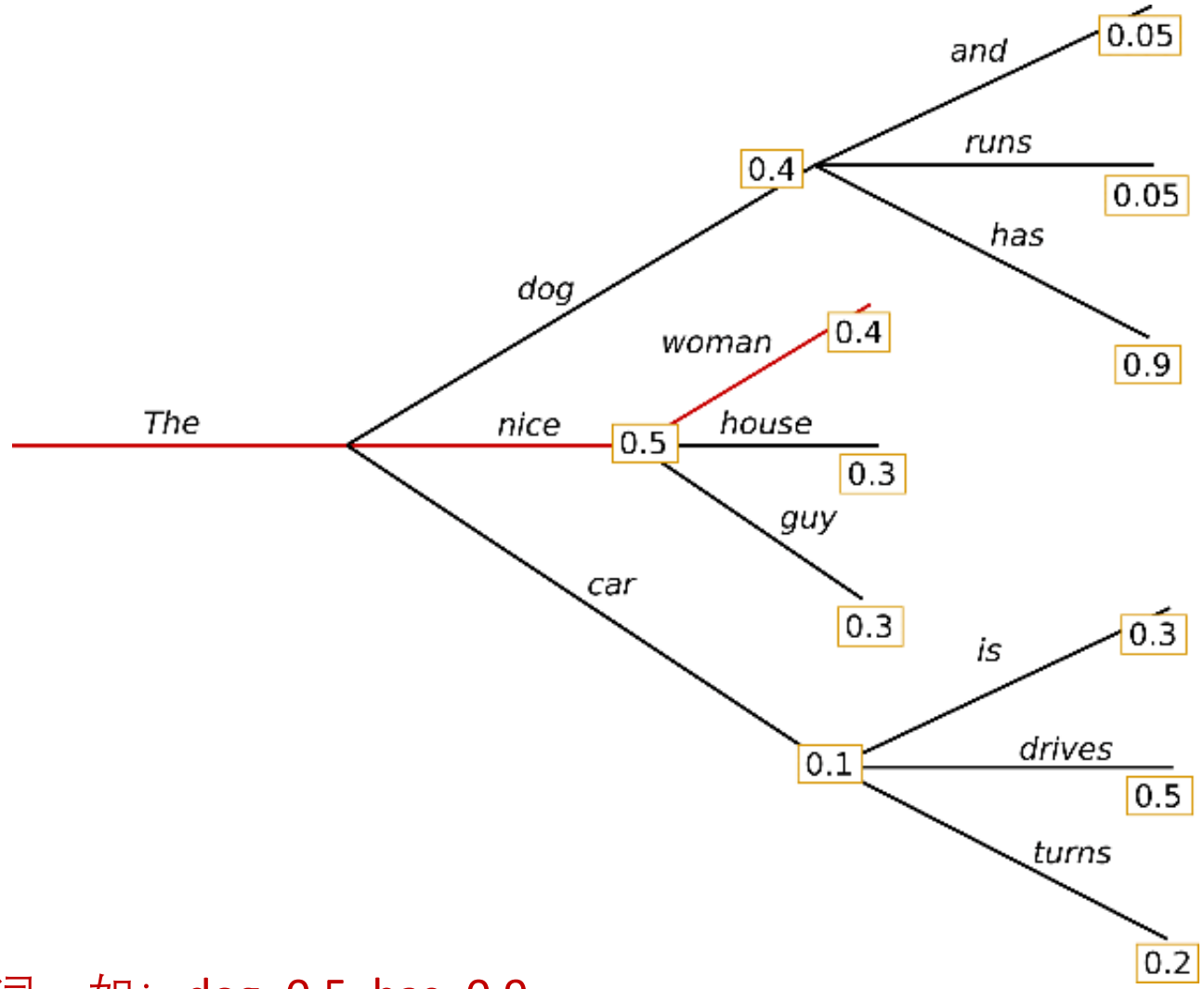
在每个时间步t都简单地选择概率最高的词作为当前输出词:

$$w_t = argmax_w P(w|w_{1:t-1})$$

按照贪心搜索输出序列

("The","nice","woman") 的条件概率为:

$$0.5 \times 0.4 = 0.2$$



缺点: 错过了隐藏在低概率词后面的高概率词，如：dog=0.5, has=0.9

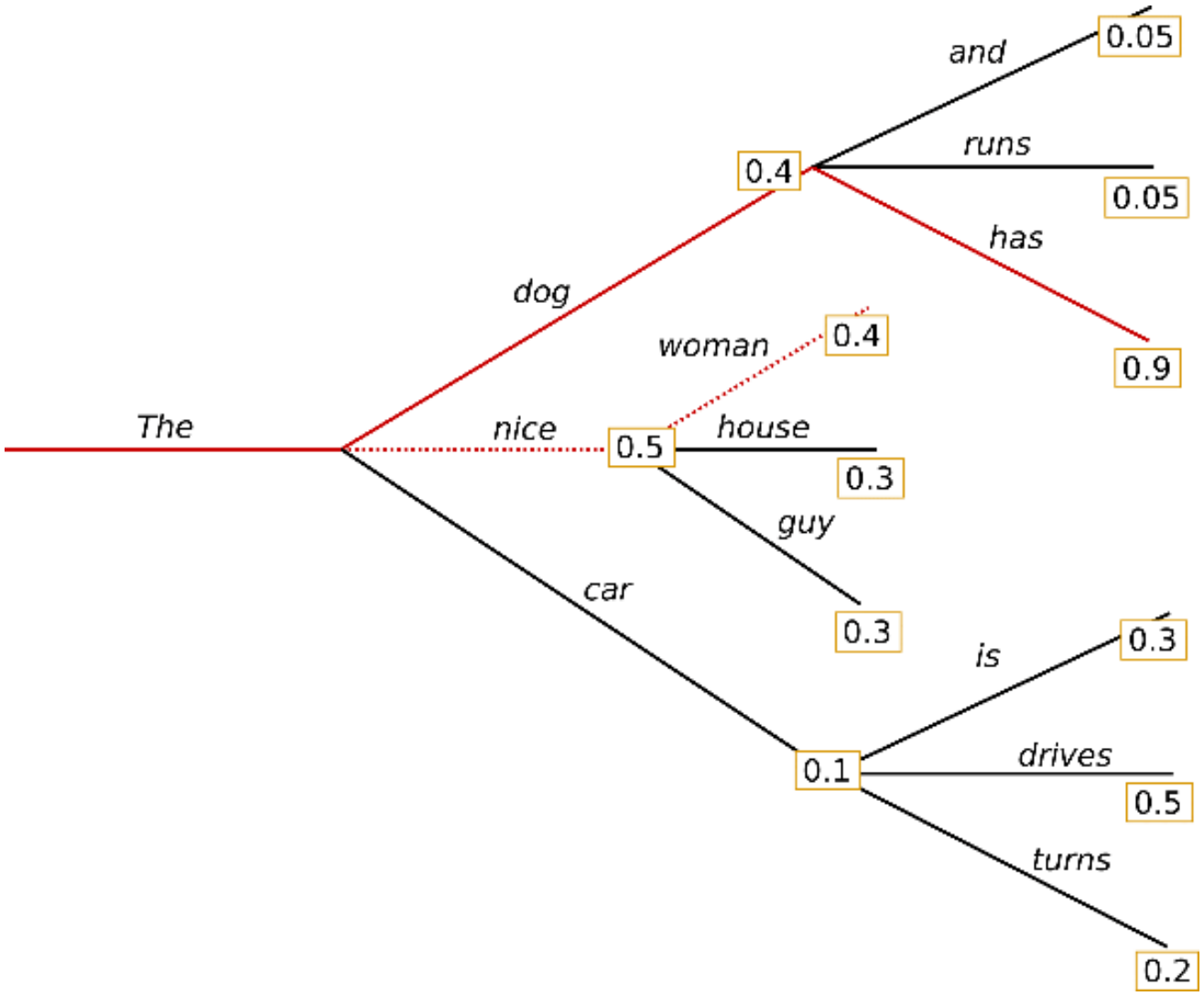
Code Example

```python
from mindnlp.transformers import GPT2Tokenizer, GPT2LMHeadModel

tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

# add the EOS token as PAD token to avoid warnings
model = GPT2LMHeadModel.from_pretrained("gpt2", pad_token_id=tokenizer.eos_token_id)

# encode context the generation is conditioned on
input_ids = tokenizer.encode('I enjoy walking with my cute dog', return_tensors='ms')

# generate text until the output length (which includes the context length) reaches 50
greedy_output = model.generate(input_ids, max_length=50)

print("Output:\n" + 100 * '-')
print(tokenizer.decode(greedy_output[0], skip_special_tokens=True))
```

Beam search

Beam search通过在每个时间步保留最可能的 num_beams 个词，并从中最终选择出概率最高的序列来降低丢失潜在的高概率序列的风险。右图以 num_beams=2 为例:

- ("The","dog","has") : 0.4 * 0.9 = 0.36
- ("The","nice","woman") : 0.5 * 0.4 = 0.20



优点：一定程度保留最优路径

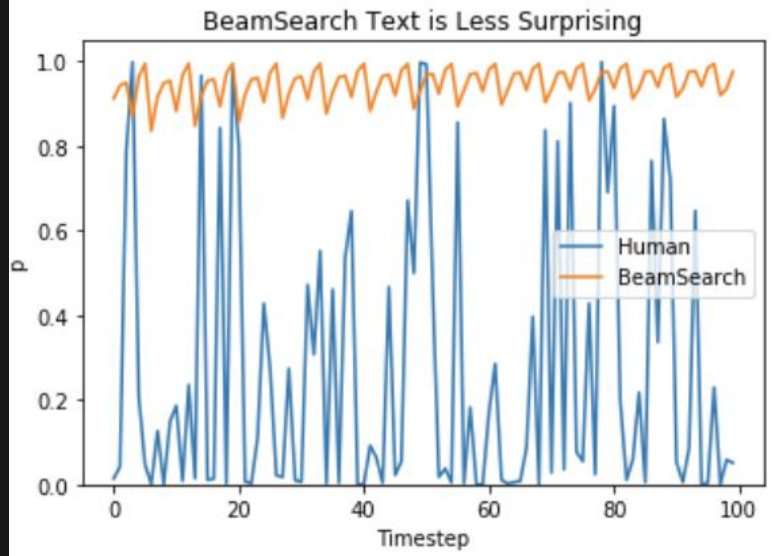
缺点：1. 无法解决重复问题；2. 开放域生成效果差

Code Example

```python
from mindnlp.transformers import GPT2Tokenizer, GPT2LMHeadModel

tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

# add the EOS token as PAD token to avoid warnings
model = GPT2LMHeadModel.from_pretrained("gpt2", pad_token_id=tokenizer.eos_token_id)

# encode context the generation is conditioned on
input_ids = tokenizer.encode('I enjoy walking with my cute dog', return_tensors='ms')

# activate beam search and early_stopping
beam_output = model.generate(
    input_ids,
    max_length=50,
    num_beams=5,
    early_stopping=True
)

print("Output:\n" + 100 * '-')
print(tokenizer.decode(beam_output[0], skip_special_tokens=True))
print(100 * '-')
```

Beam search issues

```
Output:
----------------------------------------------------------------------------------------------------
I enjoy walking with my cute dog, but I'm not sure if I'll ever be able to walk with him again.

I'm not sure if I'll ever be able to walk with him again. I'm not sure if I'll
----------------------------------------------------------------------------------------------------
Beam search with ngram, Output:
----------------------------------------------------------------------------------------------------
I enjoy walking with my cute dog, but I'm not sure if I'll ever be able to walk with him again.

I've been thinking about this for a while now, and I think it's time for me to take a break
----------------------------------------------------------------------------------------------------
return_num_sequences, Output:
----------------------------------------------------------------------------------------------------
0: I enjoy walking with my cute dog, but I'm not sure if I'll ever be able to walk with him again.

I've been thinking about this for a while now, and I think it's time for me to take a break
1: I enjoy walking with my cute dog, but I'm not sure if I'll ever be able to walk with him again.

I've been thinking about this for a while now, and I think it's time for me to get back to
2: I enjoy walking with my cute dog, but I'm not sure if I'll ever be able to walk with her again.

I've been thinking about this for a while now, and I think it's time for me to take a break
3: I enjoy walking with my cute dog, but I'm not sure if I'll ever be able to walk with her again.

I've been thinking about this for a while now, and I think it's time for me to get back to
4: I enjoy walking with my cute dog, but I'm not sure if I'll ever be able to walk with him again.

I've been thinking about this for a while now, and I think it's time for me to take a step
----------------------------------------------------------------------------------------------------
```



缺点：1. 无法解决重复问题；2. 开放域生成效果差

Repeat problem

```
Output:
----------------------------------------------------------------------------------------------------
I enjoy walking with my cute dog, but I'm not sure if I'll ever be able to walk with him again.

I'm not sure if I'll ever be able to walk with him again. I'm not sure if I'll
----------------------------------------------------------------------------------------------------
```

```
# set no_repeat_ngram_size to 2
beam_output = model.generate(
    input_ids,
    max_length=50,
    num_beams=5,
    no_repeat_ngram_size=2,
    early_stopping=True
)
```

n-gram 惩罚:

将出现过的候选词的概率设置为 0

设置no_repeat_ngram_size=2，任意 *2-gram* 不会出现两次

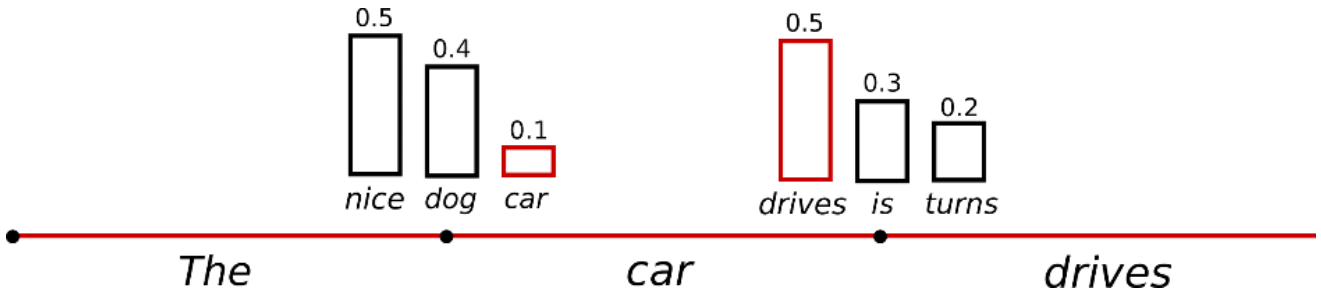
Notice: 实际文本生成需要重复出现

Sample

根据当前条件概率分布随机选择输出词w_t：

$$w_t \sim P(w|w_{1:t-1})$$

- ("car") ~ *P*(*w*|"The")
- ("drives") ~ *P*(*w*|"The","car")



优点：文本生成多样性高

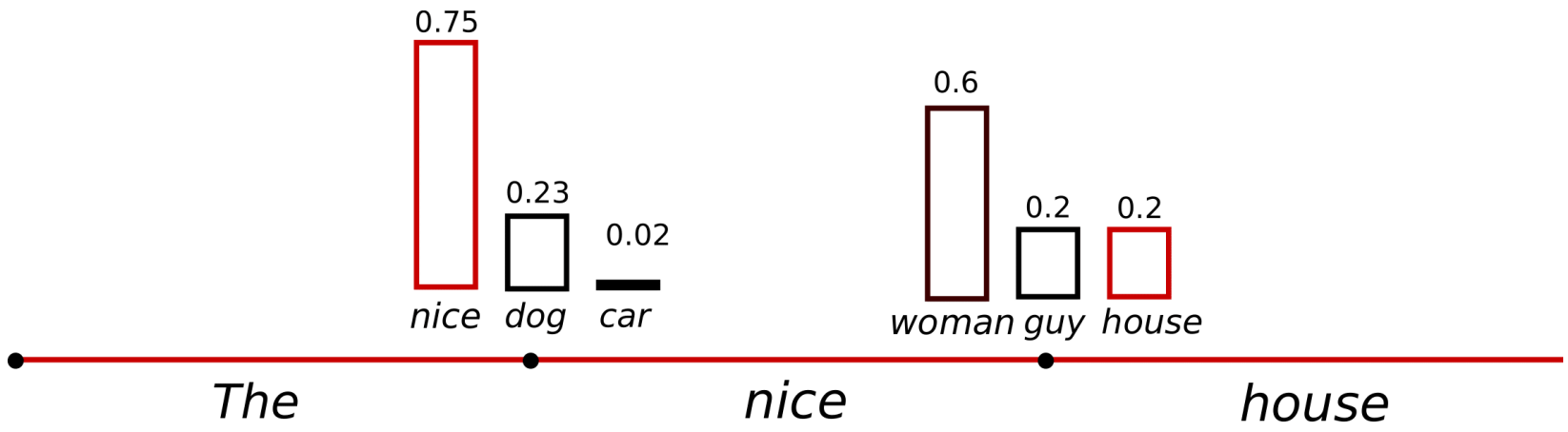
缺点：生成文本不连续

Code Example

```python
from mindnlp.transformers import GPT2Tokenizer, GPT2LMHeadModel

tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

# add the EOS token as PAD token to avoid warnings
model = GPT2LMHeadModel.from_pretrained("gpt2", pad_token_id=tokenizer.eos_token_id)

# encode context the generation is conditioned on
input_ids = tokenizer.encode('I enjoy walking with my cute dog', return_tensors='ms')

# activate sampling and deactivate top_k by setting top_k sampling to 0
sample_output = model.generate(
    input_ids,
    do_sample=True,
    max_length=50,
    top_k=0
)

print("Output:\n" + 100 * '-')
print(tokenizer.decode(sample_output[0], skip_special_tokens=True))
```

Temperature

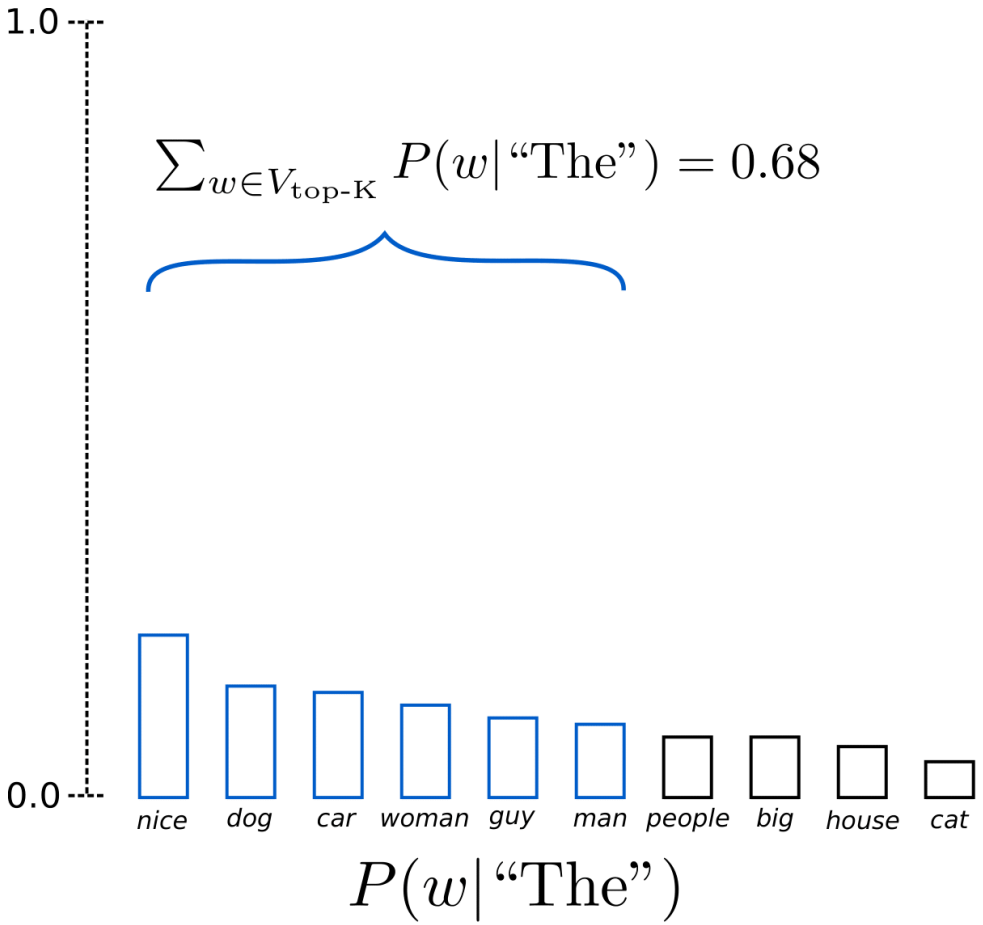
降低softmax 的temperature使 $P(w \mid w_{1:t-1})$分布更陡峭

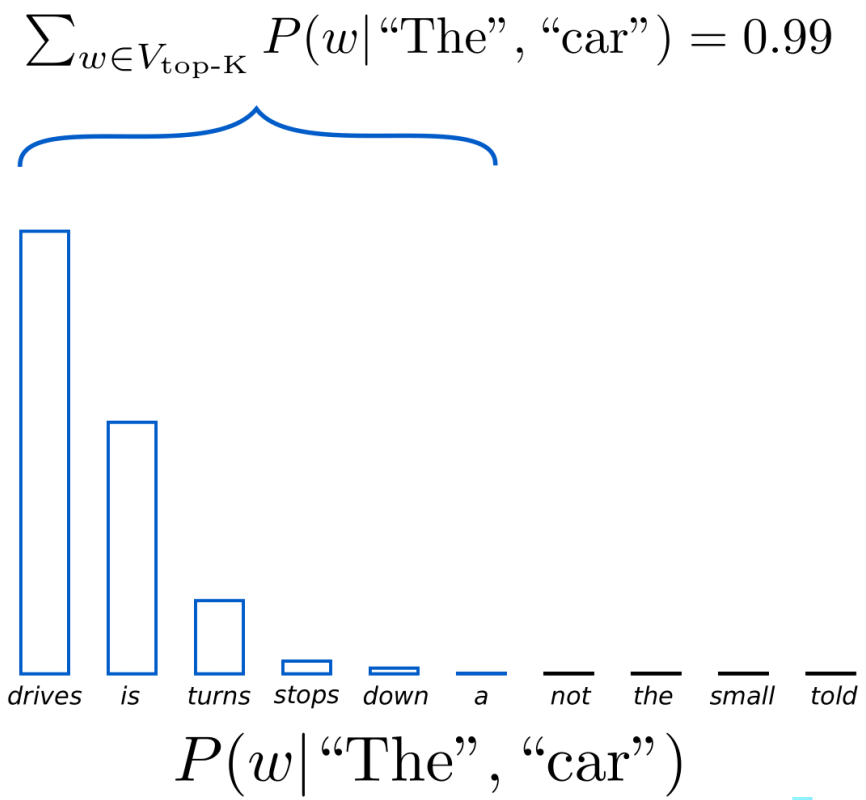


增加高概率单词的似然并降低低概率单词的似然

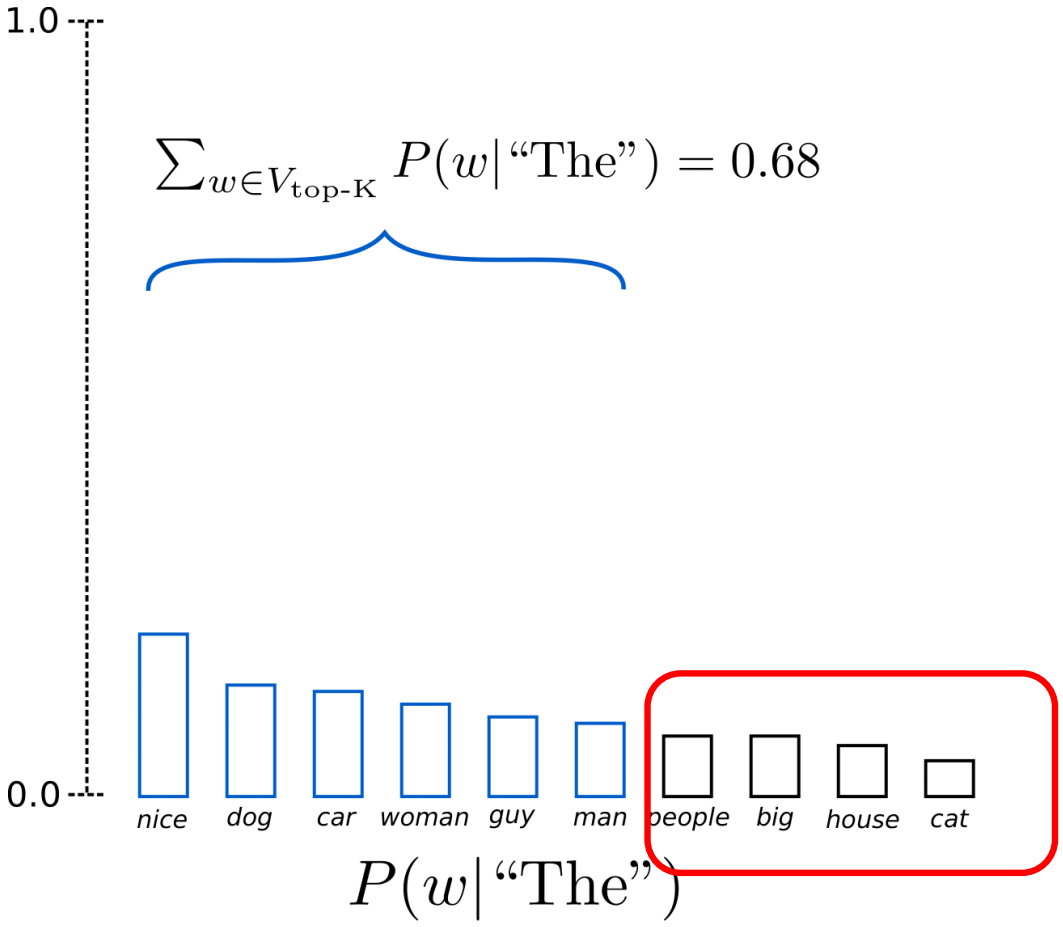
TopK sample

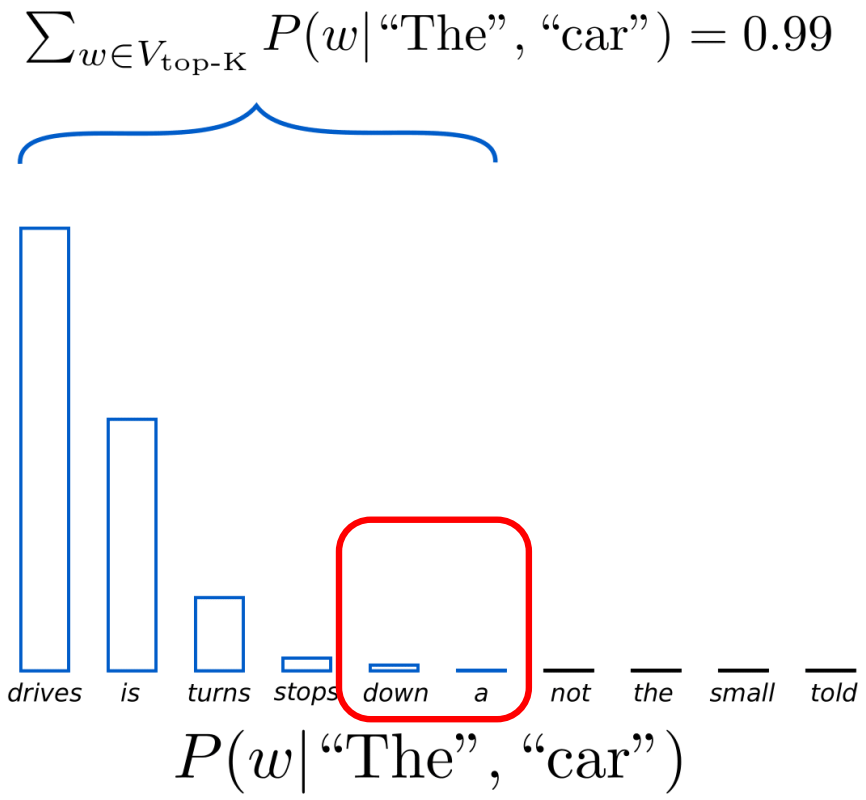
选出概率最大的 K 个词，重新归一化，最后在归一化后的 K 个词中采样





TopK sample problems



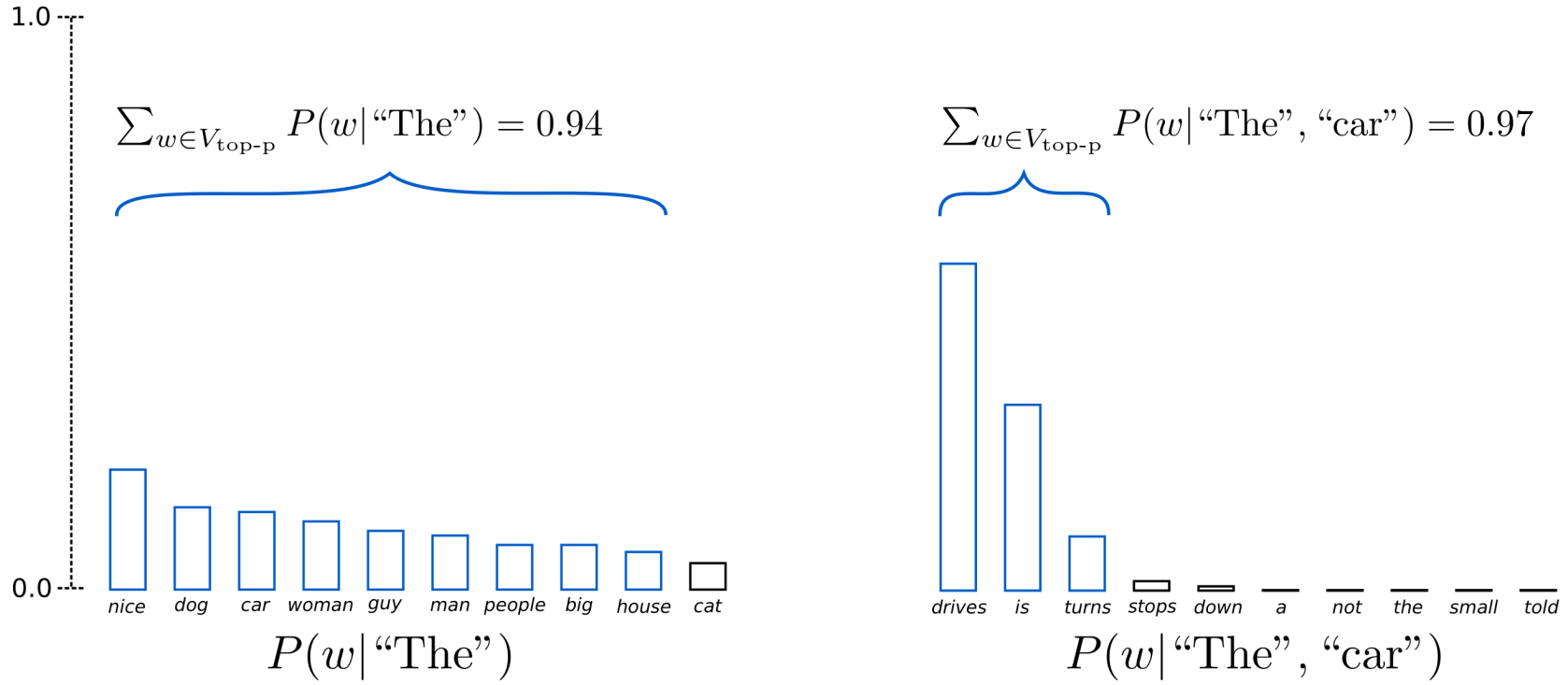


将采样池限制为固定大小 K :

- 在分布比较尖锐的时候产生胡言乱语
- 在分布比较平坦的时候限制模型的创造力

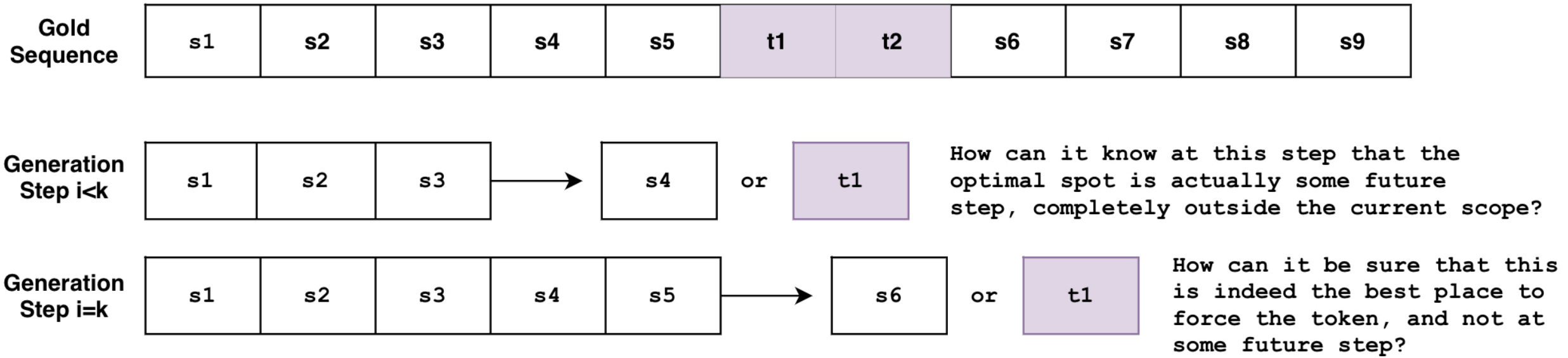
Top-P sample

在累积概率超过概率 *p* 的最小单词集中进行采样，重新归一化

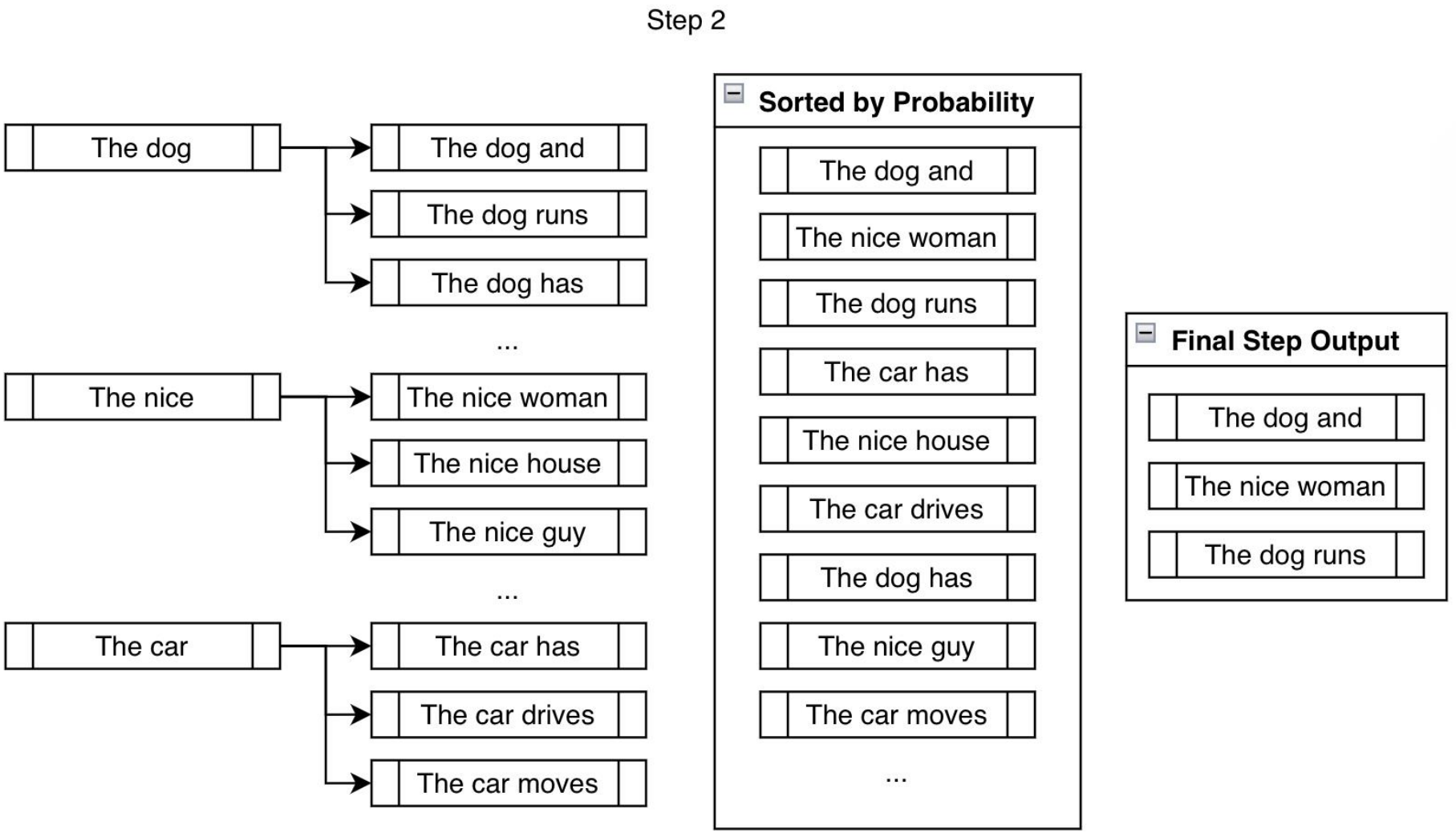


采样池可以根据下一个词的概率分布动态增加和减少

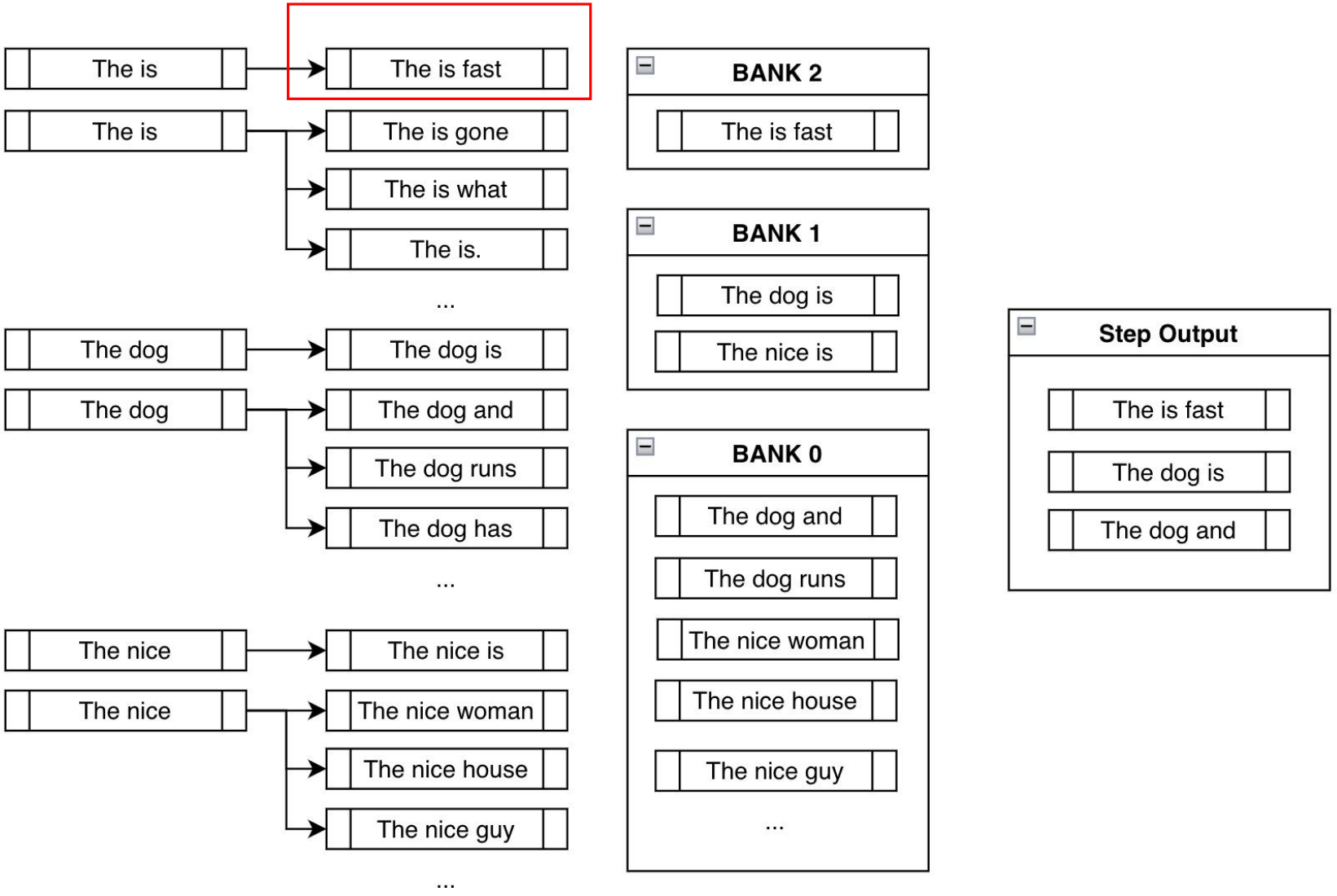
指定文本生成内容



Beam search


Step 2
The dog
The dog and
The dog runs
The dog has
...
The nice
The nice woman
The nice house
The nice guy
...
The car
The car has
The car drives
The car moves
Sorted by Probability
The dog and
The nice woman
The dog runs
The car has
The nice house
The car drives
The dog has
The nice guy
The car moves
...
Final Step Output
The dog and
The nice woman
The dog runs

Constrained Beam search



现有解码方法对比

- 确定性方法：
  - Greedy search
  - Beam search

问题：生成的文本不自然且包含不必要的重复

- 不确定性方法：
  - Top-k
  - Top-p

问题：生成文本的语义一致性问题

Contrastive Search

给定前缀文本$x_{<t}$，按如下公式选择输出Token x_t:

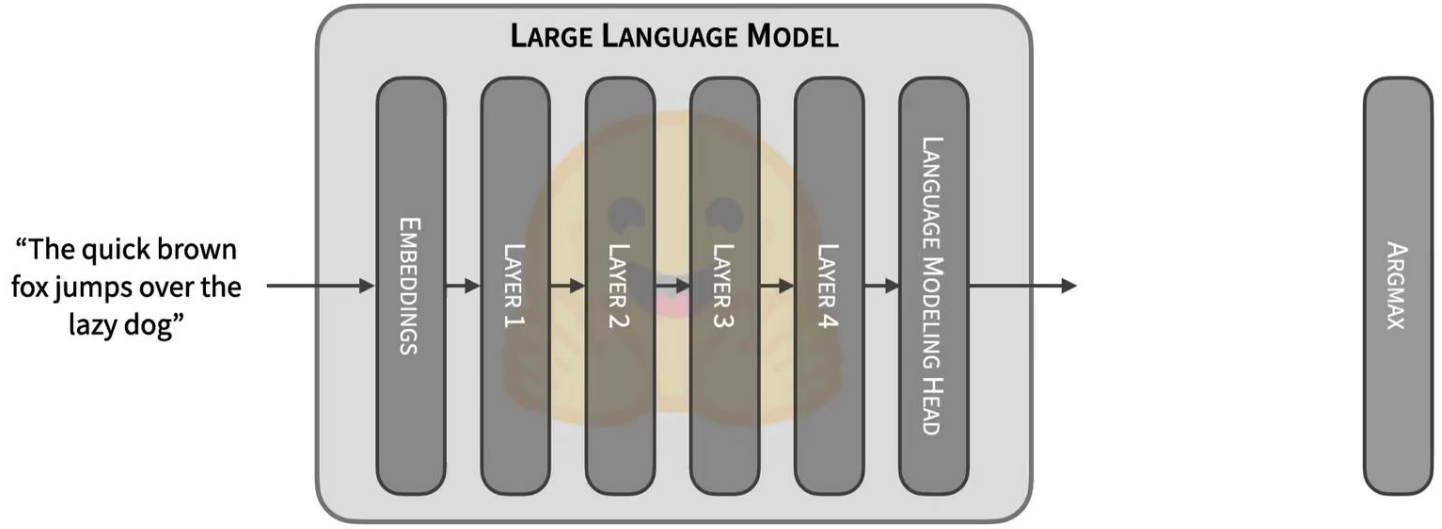
$$x_t = \arg\max_{v \in V^{(k)}} \left\{ (1-\alpha) \times \underbrace{p_\theta(v|\boldsymbol{x}_{<t})}_{\text{model confidence}} - \alpha \times \underbrace{(\max\{s(h_v, h_{x_j}) : 1 \le j \le t-1\})}_{\text{degeneration penalty}} \right\},$$

- $V^{(k)}$：k 个概率最大的候选词元的集合
- model confidence：语言模型预测的每个候选token v的概率
- degeneration penalty：用于衡量 v与上文$x_{<t}$中每个token的差异度，其中函数$s()$用于计算每两个token间的余弦相似度

如果 v的退化惩罚较大意味着它与上文更相似，因此更有可能导致模型退化问题。

LLM快速文本生成

预测辅助模型



预测辅助模型