arXiv:2603.05193v1 [cs.CL] 5 Mar 2026

转换语言模型



VésteinnSnæbjarnarson[*♀, ♈] **Samuel Kiegeland**[*♀, ♑] **Tianyu Liu**[♀] **RedaBoumasmoud**[♀] **RyanCotterell**[♀] **TimVieira**[♀♀]苏黎世联邦理工学院 [♈]哥本哈根大学 [♑]
CHI-FRO{vest.snae, tim.f.vieira}@gmail.com
reda.boumasmoud@math.ethz.ch{samuel.kiegeland, tianyu.liu, ryan.cotterell}@inf.ethz.ch

摘要

现代语言模型定义了字符串上的分布，但下游任务往往需要不同的输出格式。例如，生成字节对字符串的模型不会直接产生词级预测，而 DNA模型也不会直接产生氨基酸序列。在这种情况下，确定性字符串到字符串的变换可以将模型输出转换为所需形式。这是概率论中的一种常见模式：对随机变量 $X \sim p$ 应用函数f，会得到一个变换后的随机变量 $f(X)$，并由此诱导出一个分布。尽管语言建模中偶尔会使用此类变换，但以往的研究并未将其视为能够产生新的、完全函数式语言模型的方法。我们将这一视角形式化，并提出一个通用框架，用于构建由确定性字符串到字符串变换派生的语言模型。我们聚焦于可表示为有限状态转换器的变换——有限状态转换器是一种常用的状态机抽象，可高效实现字符串到字符串映射。我们开发了将语言模型与有限状态转换器组合的算法，以对映射到给定目标的源字符串进行边缘化，在不改变模型参数的情况下通过转换器传播概率，并支持对变换后的输出进行条件化。我们提出了一种精确算法、一种高效近似方法，并进行了理论分析。我们在三个领域开展实验：将语言模型从词元转换为字节、从词元转换为单词，以及从 DNA 转换为氨基酸。这些实验展示了如何在推理时适配预训练语言模型，使其符合特定应用的输出要求。

https://github.com/rycolab/transducing-language-models

1 引言

语言模型（LM）定义了字符串上的分布。然而，它们生成的字符串往往不符合下游应用的要求，因此从业者不得不采用临时的后处理方法。我们将此称为**字符串不匹配问题**。例如，在自然语言处理中，现代语言模型通常会生成字节对编码字符串（Sennrich 等人，2016年），而下游任务可能需要单词或字符（见下文例(2)）。类似地，DNA语言模型生成核碱基序列，而许多应用需要氨基酸序列（例(5)）。

在生成流程中加入字符串到字符串的变换，是一种常见的工程解决方案，例如对输出进行规范化，或将子词词元映射为字节。形式上，这定义了一个作用于变换后字符串的新语言模型。然而，尽管采样仍然简单明了，其他操作——例如计算变换后字符串的概率，或以变换后的输出为条件进行条件化——则变得难以处理。例如，考虑将任意大小写形式的字符串映射为其小写形式，如图1所示。将给定输入转为小写轻而易举，但将原始分布转换为小写单词上的分布却并非如此。

本研究将字符串到字符串变换视为语言建模流水线中的一等组件。我们展示了如何为这些变换后的模型配备熟悉的自回归接口——增量式的下一符号分布和前缀概率——使其能够与

*贡献相同。

任何为标准自回归语言模型构建的系统互操作。这种解决字符串不匹配问题的方法具有原则性和模块化特征，而且成本可能大幅低于重新训练。此外，这些变换通常能够保证满足下游应用的要求。

考虑英语文本上的语言模型——同一话语可以用多种方式编码。例如，

(1) *Lemaître*博士惊呆了。 🤯

GPT-4o 使用的字节对编码（OpenAI，2024）将例（(1)）编码为以下子词词元字符串：

(2)
```
Dr   .  ␣L  ema  \C3\AEtre  ␣was  ␣fl  ab
5822 13 451 4603  29135      673   1548 378
berg asted  .  ␣\F0\9F\A4  \AF
9667 23030  13  93643      107
```

例（(1)）的字节对分词基于字符子串频率。然而，许多应用需要不同的单位。例如，计算心理语言学（如Giulianelli 等人，2024）和受控生成（如Lew 等人，2023；Xefteri 等人，2025年）都需要自定义单位。如果希望推导单词上的分布，情况也同样如此，例如宾州树库（PTB）标注指南（Marcus 等人，1993）所定义的分布；其一种变体如下所示：



图1：GPT-2对BPE词元字符串在转为小写后与你好匹配的概率。所有此类序列的总概率位于底部。

(3) Lemaître博士惊呆了 . 🤯

对于其他应用，例如拼写纠错，我们可能也希望使用字符或 UTF-8字节表示的字符串来表示例（(2)），如下所示：[1]

(4) Lemaître博士目瞪口呆。␣\F0\9F\A4\AF 68 114 46 32 76 101 109 97 195 174 116 114 101 32 11997 115 32 102 108 97 98 98 101 114 103 97 115 116 101 100 46 32 240 159 164 175

在遗传学中，我们还有另一个表示方式各异的例子。考虑例(5)中给出的DNA序列。该序列是众多可翻译为激素催产素的序列之一，通常写作例(6)中的氨基酸序列，如下所示：

(5) T G T T A C A T A C A A A A T T G T C C T C T A G G T (6) C Y I Q N C P L G

对语言模型进行变换通常并非易事。即便是具有简单通俗描述的变换，应用起来也可能很困难。转换为字节（例(4)）可使用查找表完成，该查找表指定子词词元所映射到的字节序列。类似地，从DNA到蛋白质序列的转换（例(5)）依赖于从三字母DNA碱基序列到单个氨基酸的映射。语法词切分（例(3)）则不那么容易描述；它需要前瞻 来判断例如标点符号应否单独存在、是否属于‘Dr.’头衔的一部分，或是小数‘3.50’。仅靠简单规则并不能确保精确转换。尽管看似简单，字节变换与氨基酸转换并不直截了当：映射到每个输出序列的词元序列数量会随输出长度呈指数增长，而恰当的语言模型变换必须考虑所有源序列概率。[2]小数与头衔的例子凸显了另一层复杂性，其中变换规则需要仔细分析周围上下文。因此，近期论文聚焦于这些变换特定子集的实用方法，尤其是从子词模型估计字节级概率（Phan 等人，2024；Vieira 等人，2025a；Hayase 等人，2025年）。[3]我们将这些方法加以推广，以处理上文给出的所有例子。

本文介绍了一种使用有限状态转换器（FST）的字符串到字符串转换框架。FST 能够编码一类强大且易于处理的关系，包括上述提到的关系。我们将.....

[1] 请注意，UTF-8 允许同一字符串采用多种编码方式。例如，字符 Ç 可以编码为字节序列 `C3\A7` 或 `43\CC\A7`（`i\ÇC\82`）。参见 https://unicode.org/reports/tr15/。

[2] 图中也能看到这一点：给定输出前缀的输入前缀数量呈指数级增长：输出中的每增加一个符号，源序列的数量就会翻倍。

[3] 我们的方法与 Vieira 等人（2025a）的方法一样，该部分基于严格前缀单调变换(§6)，其中包括子词到字节的转换。

提出使用转换器对预训练语言模型进行编码，以表示这类变换，并将这些复合称为转导语言模型。有限状态转换器提供了明确的结构，可追踪原始模型中的概率应如何映射到输出序列。这使我们能够开发精确和近似算法，从而高效地对转换后的字符串进行采样、评分和条件化，且无需修改底层语言模型。我们给出了可实现精确变换的充分条件（§6），并在精确变换不可行时提供近似方法（§5）。

为验证我们的方法，我们针对上述三种应用场景构建有限状态转换器（FST）：（i）将词元转换为字节；（ii）按照宾州树库分词器插入正字法边界；（iii）将DNA序列转换为氨基酸序列。随后，我们在有限状态转换器的输入单位上使用常用的预训练语言模型，并将它们与有限状态转换器组合，以获得针对输出词元的语言模型。最后，我们使用这些设置对理论和算法方面的贡献进行基准测试。特别是，我们发现，采用一种实用的近似方法，就足以在计算成本仅为一小部分的情况下获得准确的估计。

2 背景[4]

字符串。 设 $\mathcal{X}$ 为一个**字母表**（即一个有限、非空集合）。令 $\mathcal{X}^*$ 表示由 $\mathcal{X}$ 构成的所有有限字符串（包括空字符串）的集合。如果不会与其他字母表混淆，我们就简写为 ε。我们用 $\boldsymbol{x}$, $\boldsymbol{x}' \in \mathcal{X}^*$ 表示字符串，用 $Z, Z' \subseteq \mathcal{X}^*$ 表示字符串集合。令 $\boldsymbol{x}\boldsymbol{x}'$ 表示**连接**。类似地，我们将字符串集合的连接定义为 $ZZ' \stackrel{\text{def}}{=} \{\boldsymbol{x}\boldsymbol{x}' \mid \boldsymbol{x} \in Z, \boldsymbol{x}' \in Z'\}$；在单元素集合的情形下，定义为 $\boldsymbol{x}Z' \stackrel{\text{def}}{=} \{\boldsymbol{x}\}Z'$ 和 $Z\boldsymbol{x}' \stackrel{\text{def}}{=} Z\{\boldsymbol{x}'\}$。当 $\boldsymbol{x}$ 是 $\boldsymbol{x}'$ 的**前缀** 时，我们记作 $\boldsymbol{x} \preceq \boldsymbol{x}'$；当它是**严格前缀**时，记作 $\boldsymbol{x} \prec \boldsymbol{x}'$。反过来，如果 $\boldsymbol{x} \preceq \boldsymbol{x}'$，我们称 $\boldsymbol{x}'$ 是 $\boldsymbol{x}$ 的**扩展**（当 $\boldsymbol{x} \prec \boldsymbol{x}'$ 时，称为**严格扩展**）。

语言模型。 **语言模型** $p_{\mathcal{X}}$ 是定义在一组字符串 $\mathcal{X}^*$ 上的概率分布。令 $\text{EOS} \notin \mathcal{X}$ 为一种特殊的**字符串结束符号**。我们将 $p_{\mathcal{X}}$ 的**前缀概率**定义为字符串 $X \sim p_{\mathcal{X}}$ 以给定前缀 $\boldsymbol{x}$ 开头的概率，并将**条件前缀概率**定义为：

$$\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}) \stackrel{\text{def}}{=} \sum_{\boldsymbol{x}' \in \mathcal{X}^*} p_{\mathcal{X}}(\boldsymbol{x}\boldsymbol{x}') \qquad \overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}' \mid \boldsymbol{x}) \stackrel{\text{def}}{=} \frac{\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}\boldsymbol{x}')}{\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})} \qquad \overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x}) \stackrel{\text{def}}{=} \frac{p_{\mathcal{X}}(\boldsymbol{x})}{\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})} \tag{1}$$

当 $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}) > 0$ 时；否则，我们设 $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}' \mid \boldsymbol{x})\ \text{def}= 0$，且 $\overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x})\ \text{def}= 1$。因此，对于所有 $\boldsymbol{x} \in \mathcal{X}^*$，$\overrightarrow{p_{\mathcal{X}}}(\cdot \mid \boldsymbol{x})$ 都是定义在 $\mathcal{X} \sqcup \{\text{EOS}\}$ 上的概率分布。利用这一结构，任何语言模型 $p_{\mathcal{X}}$ 都可以分解为 $p_{\mathcal{X}}(\boldsymbol{x}) = \overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x}) \prod_{t=1}^{|\boldsymbol{x}|} \overrightarrow{p_{\mathcal{X}}}(x_t \mid \boldsymbol{x}_{<t})$。这种分解定义了一种从左到右的生成过程：从 $\boldsymbol{x} = \varepsilon$ 开始，我们反复采样 $x' \sim \overrightarrow{p_{\mathcal{X}}}(\cdot \mid \boldsymbol{x})$；如果是 $x' = \text{EOS}$，就停止，否则将 $\boldsymbol{x}$ 更新为 $\boldsymbol{x}x'$。条件生成从条件前缀而非空字符串开始。我们将式（1）中的量称为语言模型 $p_{\mathcal{X}}$ 的自回归接口。

圆柱集。圆柱集是具有给定前缀的所有字符串组成的集合。令 $Z, Z' \subseteq \mathcal{X}^*$；我们将 Z 上的柱集定义为 $\langle Z \rangle\ \text{def}= Z\mathcal{X}^*$。如果 $Z = \langle Z \rangle$，我们就称 Z 为柱状。柱集的并仍然是柱集，因为柱集是向上封闭的。我们将 $\boldsymbol{x}$ 的基本圆柱定义为 $\langle \boldsymbol{x} \rangle\ \text{def}= \langle \{\boldsymbol{x}\} \rangle$。前缀基操作 $\text{pf}(Z)$ 定义为 $\text{pf}(Z)\ \text{def} = \{\boldsymbol{x} \in Z \colon \nexists \boldsymbol{x}' \in Z \colon \boldsymbol{x}' \prec \boldsymbol{x}\}$；此操作将 $\langle Z \rangle$ 唯一划分为 $\text{pf}(Z)$ 上的基本圆柱。若 $\text{pf}(Z) = Z$，我们就称 Z 为前缀无关。由于 $\text{pf}(Z)$ 是前缀无关的，基本圆柱 $\{\langle \boldsymbol{x} \rangle \mid \boldsymbol{x} \in \text{pf}(Z)\}$ 两两不相交，且 $\langle Z \rangle = \bigsqcup_{\boldsymbol{x} \in \text{pf}(Z)} \langle \boldsymbol{x} \rangle$。

转换器。转换器是一种编码字符串到字符串关系 $f \subseteq \mathcal{X}^* \times \mathcal{Y}^*$ 的状态机。当我们将 f 定义的关系表示为转换器时，就能揭示开发高效算法所需的计算结构。形式上，有限状态转换器[5]（FST）f 是一个元组 $(\mathrm{S}, \mathcal{X}, \mathcal{Y}, \mathrm{I}, \mathrm{F}, \mathrm{T})$，其中 S 是有限状态集合，$\mathcal{X}$ 和 $\mathcal{Y}$ 分别是输入符号和输出符号的字母表。集合 I、$\mathrm{F} \subseteq \mathrm{S}$ 分别是初始状态和接受状态。$\mathrm{T} \subseteq \mathrm{S} \times (\mathcal{X} \cup \{\varepsilon\}) \times (\mathcal{Y} \cup \{\varepsilon\}) \times \mathrm{S}$ 是转移集合。我们将转移 $(s, x, y, s') \in \mathrm{T}$ 表示为 $s \xrightarrow{x:y} s'$；我们称该转移扫描 x 并输出 y。对于状态 s 的所有出边，我们记作 $\mathrm{T}(s)$；对于扫描 x 的出边，我们记作 $\mathrm{T}(s, x)$。[6] 转换器 f 定义了一组路径 Π。每条

[4] §A 提供了一份符号表。[5] 有关转换器的详细论述，请参阅 Pin（2021年，第2章和第3章）。[6] 即，$\mathrm{T}(s'')\ \text{def} = \{(s \xrightarrow{x:y} s') \in \mathrm{T} \colon s = s''\}$，以及 $\mathrm{T}(s'', x'')\ \text{def} = \{(s \xrightarrow{x:y} s') \in \mathrm{T} \colon s = s'', x = x''\}$。

路径 $\boldsymbol{\pi} \in \Pi$ 都是由如下形式的转移组成的序列：$s_0 \xrightarrow{x_1:y_1} s_1 \xrightarrow{x_2:y_2} s_2 \cdots s_{N-1} \xrightarrow{x_N:y_N} s_N$。如果 $s_0 \in \mathrm{I}$ 且 $s_N \in \mathrm{F}$，我们就称 $\boldsymbol{\pi}$ 为接受路径。f 定义的关系为$[\![\mathrm{f}]\!] \stackrel{\text{def}}{=} \{(\boldsymbol{x}, \boldsymbol{y}) \mid s_0 \xrightarrow{x_1:y_1} s_1 \cdots s_{N-1} \xrightarrow{x_N:y_N} s_N \in \Pi\colon s_0 \in \mathrm{I}\ s_N \in \mathrm{F}\}$，也就是说，每条接受路径都会贡献一对扫描所得和输出的字符串（但不一定是唯一的）。当转换器的每个转移都扫描并输出相同的符号（$x = y$）时，该机器就作为有限状态自动机（接受器）运行，用于识别一种语言 $L \subseteq \mathcal{X}^*$——即至少有一条接受路径所接纳的字符串集合。一般而言，这种机器是非确定性有限自动机（NFA）；若它只有一个初始状态、没有 ε 转移，并且每个状态和输入符号至多对应一个转移，则它是确定性有限自动机（DFA）。通过确定化，任何 NFA 都可以转换为等价的 DFA（Rabin & Scott，1959年）。作为补充，§B 提供了更多背景信息。

3 转导语言模型

一个 **转导语言模型** $p_{\mathcal{Y}}$ 是将字符串到字符串的 **变换** f 应用于 $\mathcal{X}^* \to \mathcal{Y}^*$而得到的，其中该变换由转换器 f编码，输入字符串取自一个 **源语言模型** $p_{\mathcal{X}}$。形式上，若$X \sim p_{\mathcal{X}}$，则 $f(X)$ 具有如下概率质量函数：

$$p_{\mathcal{Y}}(\boldsymbol{y}) \stackrel{\text{def}}{=} \Pr_{X \sim p_{\mathcal{X}}}[\boldsymbol{y} = f(X)] = \sum_{\boldsymbol{x} \in f^{-1}(\boldsymbol{y})} p_{\mathcal{X}}(\boldsymbol{x}) \tag{2}$$

其中 $f^{-1}(\boldsymbol{y})$ 是 $\boldsymbol{y}$ 的原像，$f^{-1}(\boldsymbol{y})$ 定 $= \{\boldsymbol{x} \in \mathcal{X}^*$ 义为：$\boldsymbol{y} = f(\boldsymbol{x})\}$。换句话说，在式 (2) 中，我们对满足 $f(\boldsymbol{x}) = \boldsymbol{y}$ 的字符串 $\boldsymbol{x}$ 求和。遗憾的是，使用式 (2) 精确计算 $p_{\mathcal{Y}}(\boldsymbol{y})$ 通常不可行（因为原像 $f^{-1}(\boldsymbol{y})$ 可能非常大），尽管从 $p_{\mathcal{Y}}$ 进行精确采样很高效（通过采样 $\boldsymbol{x} \sim p_{\mathcal{X}}$ 并应用 f 即可）。与所有语言模型一样，转导语言模型 $p_{\mathcal{Y}}$ 也具有前缀概率函数和条件前缀概率函数；其前缀概率为

$$\overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y}) = \Pr_{X \sim p_{\mathcal{X}}}[\boldsymbol{y} \preceq f(X)] = \sum_{\boldsymbol{x} \in \mathcal{P}(\boldsymbol{y})} p_{\mathcal{X}}(\boldsymbol{x}) \tag{3}$$

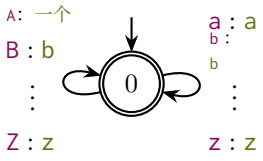
其中 $\mathcal{P}(\boldsymbol{y})$ 是 **相对于** $\boldsymbol{y}$而言的 f预覆盖，定义为 $\mathcal{P}(\boldsymbol{y}) \stackrel{\text{def}}{=} \{\boldsymbol{x} \in \mathcal{X}^*\colon \boldsymbol{y} \preceq f(\boldsymbol{x})\}$。[7]

前缀概率给出了字符串概率的条件因式分解（见 §2），从而能够高效地进行从左到右的自回归生成。我们在 §4 中提出了一种方法，使我们能够在 有限 时间内，为一类一般的映射计算公式（3）中的总和，例如引言中提到的那些映射（即文本规范化、插入正字法词边界，或将 DNA 转换为氨基酸序列）。在 §5中，我们介绍了用于计算这些量的算法。

4 分解预覆盖

在 §3中，我们看到，如果能够对 $\boldsymbol{y}$的预覆盖求和，就可以计算 $\overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y})$（公式（3）），从而为转导语言模型提供自回归接口。下面的两个示例说明，通过利用转换器的结构属性，我们通常可以计算这个无限总和。

例1. 下面的转换器会将字符串转换为小写。对于目标字符串 ab，其预覆盖是无限集合 $\mathcal{P}(\mathsf{ab}) = \langle\{\mathsf{AB}, \mathsf{Ab}, \mathsf{aB}, \mathsf{ab}\}\rangle$。给定一个输入语言上的模型 $p_{\mathcal{X}}$，右侧的推导（*4a–4c*）应用公式 (3)，将 $\overrightarrow{p_{\mathcal{Y}}}(\mathsf{ab})$ 表示为四个源语言前缀概率之和：



$$\overrightarrow{p_{\mathcal{Y}}}(\mathsf{ab}) = \sum_{\boldsymbol{x} \in \mathcal{P}(\mathsf{ab})} p_{\mathcal{X}}(\boldsymbol{x}) \tag{4a}$$

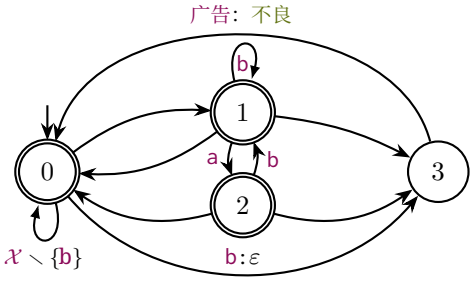
$$= \sum_{\boldsymbol{x}' \in \{\mathsf{AB},\mathsf{Ab},\mathsf{aB},\mathsf{ab}\}} \sum_{\boldsymbol{x} \in \langle \boldsymbol{x}' \rangle} p_{\mathcal{X}}(\boldsymbol{x}) \tag{4b}$$

$$= \overrightarrow{p_{\mathcal{X}}}(\mathsf{AB}) + \overrightarrow{p_{\mathcal{X}}}(\mathsf{Ab}) + \overrightarrow{p_{\mathcal{X}}}(\mathsf{aB}) + \overrightarrow{p_{\mathcal{X}}}(\mathsf{ab}) \tag{4c}$$

在例 1中，每个输入符号都恰好映射到一个输出符号，因此预覆盖可以整齐地分解为柱集。下一个例子展示了当某些源字符串覆盖目标字符串，而它们的扩展却不覆盖时会发生什么。

[7]注意，预覆盖取决于 f；当上下文已明确表明这一依赖关系时，我们会省略它。

例 2。 下面的转换器实现了一条 新话[8] 改写规则：单词 bad 被替换为 ungood。对于目标字符串 ba，预覆盖无法整齐地分解为柱集：由于bad 不会映射到以 ba 为前缀的字符串，因此柱集 $\langle \mathsf{bad} \rangle$ 不会对 $\overrightarrow{p_{\mathcal{Y}}}(\mathsf{ba})$ 作出贡献。右侧的推导（5–5c）展示了如何排除这个柱集。[9]



$$\overrightarrow{p_{\mathcal{Y}}}(\mathsf{ba}) = \sum_{\boldsymbol{x} \in \langle \mathsf{ba} \rangle \setminus \langle \mathsf{bad} \rangle} p_{\mathcal{X}}(\boldsymbol{x}) \tag{5a}$$

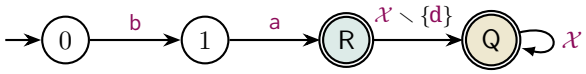
$$= \sum_{\boldsymbol{x} \in \bigcup_{x \in \mathcal{X} \setminus \{\mathsf{d}\}} \langle \mathsf{ba}x \rangle \cup \{\mathsf{ba}\}} p_{\mathcal{X}}(\boldsymbol{x}) \tag{5b}$$

$$= p_{\mathcal{X}}(\mathsf{ba}) + \sum_{x \in \mathcal{X} \setminus \{\mathsf{d}\}} \overrightarrow{p_{\mathcal{X}}}(\mathsf{ba}x) \tag{5c}$$

因为 糟糕 $\in \langle \mathsf{ba} \rangle$ 但 糟糕 $\notin \mathcal{P}(\mathsf{ba})$，预覆盖无法完全分解为柱集。相反，我们将其分解为两个不相交的部分：一个最大的柱状子集，以及它在预覆盖中的补集。商 收集每个柱集中的最短元素；补集则是余项。最后一步 (5c) 展示了一种**计算捷径**；对于任意 $\boldsymbol{y}$ 我们都可以将 $\overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y})$分解为：

$$\overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y}) = \underbrace{\sum_{\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y})} \overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})}_{\text{Quotient}} + \underbrace{\sum_{\boldsymbol{x} \in \mathcal{R}(\boldsymbol{y})} p_{\mathcal{X}}(\boldsymbol{x})}_{\text{Remainder}} \tag{6}$$

预覆盖也可以表示为有限状态自动机，如下所示，针对 $\mathcal{P}(\mathsf{ba})$。在 R (青绿色 节点) 接受的唯一字符串标记余项元素 ba，而状态 Q (黄色 节点) 接受$\langle \mathsf{ba} \cdot (\mathcal{X} \setminus \{\mathsf{d}\}) \rangle$，并标记商上的柱集。



We 现在，我们将对任意字符串到字符串函数 f的余项、商和分解进行形式化定义。

预覆盖的前缀分解。设 $f\colon \mathcal{X}^* \to \mathcal{Y}^*$ 是一个映射。对每个 $\boldsymbol{y} \in \mathcal{Y}^*$，定义 $\mathcal{C}(\boldsymbol{y})$ def $= \{\boldsymbol{x} \in \mathcal{X}^* \colon \langle \boldsymbol{x} \rangle \subseteq \mathcal{P}(\boldsymbol{y})\}$；此集合是一个柱集——若 $\boldsymbol{x} \in \mathcal{C}(\boldsymbol{y})$，则 $\boldsymbol{x}$ 的每个扩展也都属于 $\mathcal{C}(\boldsymbol{y})$——因此它是包含在 $\mathcal{P}(\boldsymbol{y})$ 中的最大柱集。我们将 $\boldsymbol{y}$ 相对于 f 的商与余项定义为

$$\mathcal{Q}(\boldsymbol{y}) \stackrel{\text{def}}{=} \mathrm{pf}(\mathcal{C}(\boldsymbol{y})) \qquad \text{and} \qquad \mathcal{R}(\boldsymbol{y}) \stackrel{\text{def}}{=} \mathcal{P}(\boldsymbol{y}) \setminus \mathcal{C}(\boldsymbol{y}) \tag{7}$$

我们称这对 $(\mathcal{Q}(\boldsymbol{y}) \;\; \mathcal{R}(\boldsymbol{y}))$ 为 $\mathcal{P}(\boldsymbol{y})$ 的最优前缀分解，其特征是满足三个条件：(i) $\mathcal{Q}(\boldsymbol{y})$ 是前缀无关的，(ii) $\mathcal{P}(\boldsymbol{y}) = \langle \mathcal{Q}(\boldsymbol{y}) \rangle \sqcup \mathcal{R}(\boldsymbol{y})$（有效性），以及 (iii) $\langle \mathcal{Q}(\boldsymbol{y}) \rangle = \mathcal{C}(\boldsymbol{y})$（极大性）——$\mathcal{Q}(\boldsymbol{y})$ 标识了 $\mathcal{P}(\boldsymbol{y})$ 中最大的柱集。这个分解（实际上，任何有效分解）都能让我们使用以下捷径计算前缀概率：

$$\overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y}) = \sum_{\boldsymbol{x} \in \mathcal{P}(\boldsymbol{y})} p_{\mathcal{X}}(\boldsymbol{x}) = \sum_{\boldsymbol{x} \in \mathcal{C}(\boldsymbol{y}) \sqcup \mathcal{R}(\boldsymbol{y})} p_{\mathcal{X}}(\boldsymbol{x}) = \sum_{\substack{\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y}) \\ \boldsymbol{x}' \in \mathcal{X}^*}} p_{\mathcal{X}}(\boldsymbol{x}\boldsymbol{x}') + \sum_{\boldsymbol{x} \in \mathcal{R}(\boldsymbol{y})} p_{\mathcal{X}}(\boldsymbol{x}) = \sum_{\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y})} \overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}) + \sum_{\boldsymbol{x} \in \mathcal{R}(\boldsymbol{y})} p_{\mathcal{X}}(\boldsymbol{x}) \tag{8a}$$

§5 给出了计算前缀分解的算法；§6 指出其何时是有限的 .

5 算法

现在，我们通过利用编码该函数的转换器的显式结构，提出一种计算前缀分解的算法。结合计算捷径（公式（6）），该算法为转换语言模型提供了一个自回归接口。我们首先抽象地描述该算法，然后使用转换器实现其中的检查，最后讨论优化。

[8]新话是奥威尔的《一九八四》中的受控语言《一九八四》(奥威尔，1949)；示例坏 → 不佳是规范形式。完整的新话转换器留作党的一道练习题。[9]由单个符号标记的弧称为复制转移，例如，b 是b : b的简写；像 $\mathcal{X} \setminus \{\mathsf{b}\}$ 这样的集合标签表示集合中每个符号各有一条复制转移弧。

图 2 给出了分解算法（decompose）。该算法维护一个候选源字符串队列，并通过广度优先搜索（广度优先搜索）探索这些字符串，还可以选择在每个步数剪枝低概率候选项（如下所述）。[10] 每个从队列中取出的字符串 $\boldsymbol{x}$ 都会接受三项检查，这些检查根据预覆盖 $\mathcal{P}(\boldsymbol{y})$定义：

1. 柱集：is_cylinder($\boldsymbol{x}, \boldsymbol{y}$) $\iff \langle \boldsymbol{x} \rangle \subseteq \mathcal{P}(\boldsymbol{y})$，即 $\boldsymbol{x}$ 的每个扩展都覆盖 $\boldsymbol{y}$。$\boldsymbol{x}$ 会被加入商集 Q，且不再继续探索（第 12 行）。2. 成员：is_member($\boldsymbol{x}, \boldsymbol{y}$) $\iff \boldsymbol{x} \in \mathcal{P}(\boldsymbol{y})$，即 $\boldsymbol{x}$ 本身覆盖 $\boldsymbol{y}$。当 is_cylinder 为假但 is_member 为真时，$\boldsymbol{x}$ 会被加入余项集 R；其扩展仍会继续探索（第 15 行）。3. 活性：is_live($\boldsymbol{x}, \boldsymbol{y}$) $\iff \exists \boldsymbol{x}'' \in \mathcal{X}^*$：$\boldsymbol{x}\boldsymbol{x}'' \in \mathcal{P}(\boldsymbol{y})$，即 $\boldsymbol{x}$ 的某个扩展属于预覆盖。只有活性扩展才会入队（第 18 行）。

由于字符串按长度从短到长处理，如果当前字符串的任何前缀已进入商集，那么当前字符串就不会入队。队列耗尽后，算法返回前缀分解。我们在 §5.1 中具体实例化这些检查。

```
1  def decompose(y): # memoize
2    N ← |y|
3    if N = 0:
4      q ← QUEUE({ε})
5    else:
6      (Q′, R′) ← decompose(y_{<N})
7      q ← QUEUE(Q′ ∪ R′)
8    (Q, R) ← (∅, ∅)
9    while |q| > 0:
10     q′ ← ∅
11     for x ∈ q:
12       if is_cylinder(x, y):
13         Q.add(x)
14         continue
15       if is_member(x, y):
16         R.add(x)
17       for x′ ∈ X:
18         if is_live(xx′, y):
19           q′.add(xx′)
20     q ← prune(q′)
21   return (Q, R)
```

图2：分解算法。

定理5.1（分解的正确性）。 如果 $\mathcal{P}(\boldsymbol{y})$ 满足有限分解，且这三项检查在不进行剪枝的情况下精确实现了上述条件，那么 分解($\boldsymbol{y}$) 会终止，其输出 (Q, R) 就是最优前缀分解（式 (7)）。

证明。 我们验证最优前缀分解（式（7））的三个条件。

1. 前缀无关性 (Q 是前缀无关的)：当 is_柱面 检查对 $\boldsymbol{x}$ （第 12 行）成功时，continue（第 14 行）会跳过扩展循环，因此不会将任何 $\boldsymbol{x}x'$ 加入队列。由于字符串只有在出队字符串的单符号扩展时才会进入队列，因此 $\boldsymbol{x}$ 的任何扩展都不会入队或出队，所以Q 中的任意两个元素都没有共同前缀。

2. 有效性（$\mathcal{P}(\boldsymbol{y}) = \langle \mathrm{Q} \rangle \sqcup \mathrm{R}$）：令 $\boldsymbol{x} \in \mathcal{P}(\boldsymbol{y})$；我们先证明 $\boldsymbol{x} \in \langle \mathrm{Q} \rangle \sqcup \mathrm{R}$。我们用前缀长度归纳证明：要么 $\boldsymbol{x}$ 的某个前缀进入 Q，要么 $\boldsymbol{x}$ 本身出队并被加入 Q 或 R。基础情形成立：ε 入队。若 $\boldsymbol{x}_{<k}$ 出队后未被放入 Q，则扩展循环会将 $\boldsymbol{x}_{<k+1}$ 入队：因为 $\boldsymbol{x}_{<k+1} \preceq \boldsymbol{x}$ 且 $\boldsymbol{x} \in \mathcal{P}(\boldsymbol{y})$，所以有 is_活性($\boldsymbol{x}_{<k+1}, \boldsymbol{y}$)（第 18 行）。根据归纳法，要么某个前缀 $\boldsymbol{x}_{<j}$ 进入 Q——从而得到 $\boldsymbol{x} \in \langle \boldsymbol{x}_{<j} \rangle \subseteq \langle \mathrm{Q} \rangle$——要么 $\boldsymbol{x}$ 本身出队，柱面检查失败，且 $\boldsymbol{x} \in \mathcal{P}(\boldsymbol{y})$通过 is_成员 给出 $\boldsymbol{x} \in \mathrm{R}$（第 15 行）。反向结论显然成立，因为 Q 中的元素通过精确的 is_柱面检查，所以 $\langle \mathrm{Q} \rangle \subseteq \mathcal{P}(\boldsymbol{y})$。同理，R 中的元素通过 is_成员，因此 $\mathrm{R} \subseteq \mathcal{P}(\boldsymbol{y})$。剩下要证明的是 $\langle \mathrm{Q} \rangle$ 与 R 不相交。若 $\boldsymbol{x} \in \mathrm{R}$，则 $\boldsymbol{x}$ 已出队，且 is_柱面检查失败，因此 $\boldsymbol{x} \notin \mathrm{Q}$。$\boldsymbol{x}$ 的任何真前缀也不在 Q 中——否则 $\boldsymbol{x}$ 就不会入队（前缀无关性）。因此 $\boldsymbol{x} \notin \langle \mathrm{Q} \rangle$。

3. 极大性（$\langle \mathrm{Q} \rangle = \mathcal{C}(\boldsymbol{y})$）：最短优先处理确保商元素的任何真前缀都不满足 is_柱集（第 12 行），因此 Q 是柱集的最小前缀无关集。结合精确圆柱检查，总能找到最早符合条件的前缀，正如公式（7）所要求的那样。剩下需要证明 $\mathcal{C}(\boldsymbol{y}) \subseteq \langle \mathrm{Q} \rangle$。根据有效性，$\mathcal{P}(\boldsymbol{y}) = \langle \mathrm{Q} \rangle \sqcup \mathrm{R}$。R 中的每个元素都不满足 is_柱集，因此 $\mathrm{R} \cap \mathcal{C}(\boldsymbol{y}) = \emptyset$。所以 $\mathcal{C}(\boldsymbol{y}) \subseteq \langle \mathrm{Q} \rangle$。

终止性：is_活性 （第 18 行）确保只有可行扩展会被加入队列；分解的有限性确保队列最终清空。

■

保守检查与次优性。如果分解所用的检查是保守近似，算法可能会生成次优的分解——它们有效，但商比最优分解更小。保守的活性检查（假阳性）会将不必要的扩展加入队列，但不会影响分类：每个出队的字符串仍会被正确分类，因为是_柱集且是_成员，因此结果仍然最优。保守的柱面检查（假阴性）可能无法识别某些商元素，从而将它们放入R 中。分解

[10]为提高效率，分解应进行记忆化。对于 $|\boldsymbol{y}| > 0$，递归调用 分解（$\boldsymbol{y}_{<N}$）会使用之前分解得到的结果为队列提供初始内容 $\mathrm{Q}' \cup \mathrm{R}'$，而不是从 ε 枚举。

仍然有效，但由于成功的柱面检查会终止对该子树的探索（第14行），漏掉柱集意味着广度优先搜索会继续探索原本会被剪去的扩展，可能导致余项膨胀并增加计算量。相比之下，成员检查必须精确，才能保持有效性。

通过剪枝进行近似。当前缀分解变得庞大时，穷举枚举并评分可能变得不可行。在这种情况下，我们采用一种剪枝策略（τ），按前缀概率对候选项排序，并移除累积概率质量低于指定阈值的候选项$\langle \mathsf{Q} \rangle \sqcup \mathsf{R} \subseteq \mathcal{P}(\boldsymbol{y})$。这样可以丢弃低概率候选项，使分解保持可处理。由于剪枝只会从队列中移除候选项，因此找到的每个元素都是正确的——$\langle \mathsf{Q} \rangle \sqcup \mathsf{R} \subseteq \mathcal{P}(\boldsymbol{y})$——但通常分解不再有效（覆盖可能不完整），因此计算出的前缀概率是真实值的下界。我们在 §C.3中详细介绍了这一策略。

5.1 入门：使用预覆盖机器实例化检查

E

现在我们将展示如何使用有限状态转换器实例化这三项检查（图2）。这可作为教学导论；§C 介绍了一种更详细但更快的算法。

我们使用转换器 f 表示变换 f（参见 §2 和 § B）。给定目标前缀 $\boldsymbol{y}$，$\mathrm{proj}_{\mathcal{X}}(\mathsf{f} \circ \boldsymbol{y}\mathcal{Y}^*)$ 是一个恰好接受 $\mathcal{P}(\boldsymbol{y})$ 的 NFA。为了支持下面高效的基于状态的检查，我们将该 NFA 确定化并裁剪，从而得到一个 DFA：$\mathsf{P}_{\boldsymbol{y}} \stackrel{\text{def}}{=} \mathrm{trim}(\mathrm{determinize}(\mathrm{proj}_{\mathcal{X}}(\mathsf{f} \circ \boldsymbol{y}\mathcal{Y}^*)))$ 令 $\mathsf{S}_{\boldsymbol{y}}$、$\mathsf{I}_{\boldsymbol{y}}$、$\mathsf{F}_{\boldsymbol{y}}$ 和 $\mathsf{T}_{\boldsymbol{y}}$ 表示 $\mathsf{P}_{\boldsymbol{y}}$ 的各个组成部分。由于 $\mathsf{P}_{\boldsymbol{y}}$ 是确定性的，扫描源字符串 $\boldsymbol{x} = x_1 \cdots x_N$ 会得到唯一的状态，我们将其记为 $\mathsf{run}_{\boldsymbol{y}}(\boldsymbol{x})$。现在我们介绍如何使用 $\mathsf{P}_{\boldsymbol{y}}$ 实现这三项检查（图 3 中给出了伪代码）。

• 柱集：我们需要检查 $\langle \boldsymbol{x} \rangle \subseteq \mathcal{P}(\boldsymbol{y})$，以判断 $\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y})$。令 $S = \mathsf{run}_{\boldsymbol{y}}(\boldsymbol{x})$ 表示扫描 $\boldsymbol{x}$后到达的唯一状态。由于 $\mathsf{P}_{\boldsymbol{y}}$ 是确定性的，$\langle \boldsymbol{x} \rangle \subseteq \mathcal{P}(\boldsymbol{y})$ 当且仅当S 具有普遍性：$[\![\mathsf{P}_{\boldsymbol{y}[S]}]\!] = \mathcal{X}^*$，即 $\boldsymbol{x}$ 的每个延续都被接受。该柱面检查（图 3）通过从 S开始执行广度优先搜索（BFS）来测试普遍性。

• 成员：当是_柱集检查 $\boldsymbol{x}$ 对 $S = \mathsf{run}_{\boldsymbol{y}}(\boldsymbol{x})$失败时，我们需要确定扫描得到的字符串 $\boldsymbol{x}$ 是否属于 $\mathcal{R}(\boldsymbol{y})$。此时只需检查 $S \in \mathsf{F}_{\boldsymbol{y}}$，即检查$\mathsf{P}_{\boldsymbol{y}}$ 是否接受 $\boldsymbol{x}$。

• 活性：我们构造$\mathsf{P}_{\boldsymbol{y}}$ 作为一个修剪后的接受 $\boldsymbol{x}$ 的自动机，其中 $\boldsymbol{y} \preceq f(\boldsymbol{x})$。[11] 修剪可确保每个可达状态都位于某条接受路径上，因此，活性问题可归结为检查$\mathsf{run}_{\boldsymbol{y}}(\boldsymbol{x}) \neq \emptyset$。

虽然分解是以源字符串的形式表述的，但每次检查都会在内部计算扫描$S = \mathsf{run}_{\boldsymbol{y}}(\boldsymbol{x})$ 后到达的状态 $\boldsymbol{x}$，该过程在确定性$\mathsf{P}_{\boldsymbol{y}}$中进行，并将问题归结为状态性质。

5.2 优化

上述算法是正确的，但对于大型转换器来说并不实用。在 §C中，我们介绍了几种优化；这里概述其核心思想。

惰性确定化。对$\mathsf{P}_{\boldsymbol{y}}$ 进行前沿的提前确定化通常计算开销很大。相反，我们跟踪前沿：扫描一个源前缀后可达的转换器状态集合，以及迄今为止产生的输出。前沿以惰性方式执行子集构造（标准的 NFA 到 DFA 的转换；Rabin & Scott，1959年），并同时与 $\boldsymbol{y}\mathcal{Y}^*$ 进行复合，从而避免提前确定化和提前复合。现在，三项检查——柱集、成员和活性——均以该前沿而非单个 DFA 状态来定义（§C.2）。

增量式下一符号分解。为了高效计算 $\overrightarrow{p_{\mathcal{Y}}}(y' \mid \boldsymbol{y})$ 对所有 $y' \in \mathcal{Y}$，我们引入 分解_下一 (§C.4)，它推导出每个扩展 $\boldsymbol{y}y'$ 的分解，该分解来自 $\boldsymbol{y}$ 的分解。它在单次广度优先搜索遍历中同时处理每个 $y' \in \mathcal{Y}$。

分解捷径。分解的许多结构性质使广度优先搜索能够跳过工作：非柱面单调性（命题 C.3）、柱集唯一性（命题 C.4）、输入投影普遍性（§C.6）、组合普适性（命题C.5），以及全通用快速路径（图 14）。

更惰性的枚举。我们不显式地将转换器与复制转导器进行复合$\boldsymbol{y}\mathcal{Y}^*$，而是由基于前沿的算法直接跟踪输出缓冲区，以惰性方式执行复合，从而避免物化成本。此外，我们预先计算其输入投影

[11] This can also be expressed as $\mathcal{P}(\boldsymbol{y}) = f^{-1}(\boldsymbol{y}\mathcal{Y}^*) = [\![\mathsf{proj}_{\mathcal{X}}(\mathsf{f} \circ \boldsymbol{y}\mathcal{Y}^*)]\!]$.

```
def step_y(s, x): return T_y(s, x) # 确定性
def run_y(x_1 ··· x_N):
    if N = 0: return I_y # 唯一
    return step_y(run_y(x_<N), x_N)

def is_成员(x, y):
    return run_y(x) ∈ F_y
def is_活性(x, y): # 不处于失败状态
    return run_y(x) ≠ ∅

定义 is_柱集(x, y):
# 运行广度优先搜索以查找反例：
# 非接受状态或不完整状态
    S ← run_y(x);
    V ← {S}; q ← 队列({S})
    while q:
        S ← q.pop()
        if S ∉ F_y: return False
        for x' ∈ X:
            S' ← 步数_y(S, x')
            if S' = ∅: return False
            if S' ∉ V: q.add(S'); V.add(S')
    return True # 没有反例
```

图 3：在预覆盖 DFA 2P P_y上对图 2 中的检查进行基于状态的实例化。辅助函数 步数 和 run_成员 和 is_活性 是单状态查找；is_柱集 通过广度优先搜索验证普遍性。

接受所有 $\mathcal{X}^*$ (§C.6)。当前沿到达这样的状态，且其缓冲区已覆盖 y时，柱面检查会立即成功，从而跳过耗时的检查。

标签推送。 作为标准预处理步骤（例如参见Oncina 等人，1993；Mohri，2003；2009），我们将输出标签向 f 的初始状态推送：当经过状态 s 的每条路径发出的输出都以 y 开头时，该 y 会移至入弧，使其提前一个步数产生。这样可确保尽早提交输出，从而加快基于前沿的柱面检查。

6 有限分解的充分条件

§5中的分解算法通过广度优先搜索枚举字符串。若要使枚举终止，对于给定的目标字符串 $y \in \mathcal{Y}^*$，商 $\mathcal{Q}(y)$ 和余项 $\mathcal{R}(y)$ 必须是有限的。先前工作（Vieira 等人，2025a，命题 1 和 3；另见 §D.2）表明，严格前缀单调性可保证余项为空且商有界。这确保了 $\overrightarrow{p_{\mathcal{Y}}}(y)$ 可计算。

然而，对于非严格前缀单调的函数，分解可能是无限的。本节给出有限分解的其他条件。我们首先介绍能够保证余项为空且商有界的函数层面性质，然后给出商与余项有限时所需的转导器层面条件。

我们称映射 $f\colon \mathcal{X}^* \to \mathcal{Y}^*$ 为严格前缀单调，当且仅当 $x \prec x' \Longrightarrow f(x) \prec f(x')$。类似地，称 f 为前缀单调，当且仅当 $x \preceq x' \Longrightarrow f(x) \preceq f(x')$。我们称映射 f 为前缀连续，若对于每个 $y \in \mathcal{Y}^*$，集合 $\mathcal{P}(y)$ 都是柱状的，即 $\mathcal{R}(y) = \emptyset$。以下命题表明，前缀单调性蕴含前缀连续性：

命题 6.1. 以下各项等价：*(i)* f 是前缀单调的 *(ii)* $f(\langle x \rangle) \subseteq \langle f(x) \rangle$ 对所有 $x \in \mathcal{X}^*$ *(iii)* $\mathcal{P}(f(x)) = \mathcal{C}(f(x))$ 对所有 $x \in \mathcal{X}^*$ *(iv)* f 是前缀连续的。[证明：§D.1]

命题 6.1 展示了我们如何推广Vieira 等人 (2025a)。首先，通过将严格前缀单调性放宽为前缀单调性，我们支持商中的多符号前瞻。例 (6) 就是这样一个例子：它是前缀单调的，并且在确定输出之前需要双符号前瞻。其次，通过引入余项，我们可以处理非前缀单调的函数，这意味着可以存在覆盖目标的源字符串，但并非它们的所有扩展都能覆盖目标（例如，示例 2）。

引理6.1给出了关于转换器的充分条件，这些条件保证对每个目标字符串都存在有限分解，即使底层函数不是前缀单调的。关键概念是安全性：一个状态是安全的，如果它是 IP-通用的、具有**有限闭包**（即 $|[\![\mathsf{f}_{[s]}]\!]| < \infty$），或者它的所有后继状态都是安全的。

引理 6.1. 设 $f\colon \mathcal{X}^* \to \mathcal{Y}^*$ 是由转换器 f实现的函数。分解$(\mathcal{Q}(y)\ \mathcal{R}(y))$ 对每个 $y \in \mathcal{Y}^*$ 都是有限的，如果：

(i) 无 ε-输出环：f 中不存在每条弧都输出 ε的环。*(ii)* 安全性：f 的每个状态都是安全的，其定义为满足以下条件的最小集合：s 是安全的，当且仅当：*(a)* s是 *IP*-通用的；*(b)* $|[\![\mathsf{f}_{[s]}]\!]| < \infty$ （有限闭包）；或 *(c)* 对所有转移 $s \xrightarrow{x:y} s'$，s' 都是安全的。证明：见 §D.3。

引理 6.1 中的条件保证了精确计算。特别是，实验部分（§7）中介绍的转换器均满足这些条件：词元到字节转换器 $\mathtt{f}_\alpha$ 和 DNA 到氨基酸转换器 $\mathtt{f}_{\text{dna2aa}}$，其商有限且余项为空；但 PTB 转导器$\mathtt{f}_{\text{ptb}}$不满足这些条件，因为其商无限。

分解的有限性是函数 f的一种性质，而非对其进行编码的某个特定转换器的性质。引理中的条件是充分条件，但不是必要条件：安全性会逐一检验每个状态，而分解算法检验的是前沿（状态/字符串对的集合）的全称性；即使单个状态并不安全，这项检验仍可能通过。前缀分解可能有许多情形是无限的，或规模大到难以处理；遇到这些情况时，我们让源语言模型确定前缀分解中哪些成员承载的概率质量最显著，并剪枝其余成员（第 5 节）。这样，我们便可以通过枚举分解中概率质量较高的子集，贪心地近似转导语言模型的概率。

7 实验

现在我们考察转导语言模型的三个示例。对于每种用例，我们遵循 Vieira 等人（2025a）的方法，测量使用第 C.3 节所述的概率质量剪枝近似方法得到的分布与参考分布之间的詹森-香农散度（JSD）；参考分布通过设置剪枝阈值 τ 得到。我们还报告交叉熵损失（第 G.4 节），以反映对特定序列而非完整分布进行评分的成本。实验使用 GPT-2 Large（p_{gpt2}）（Radford 等人，2019年）、LLaMA 3.2-1B（p_{llama1B}）和 LLaMA 3.1-8B（p_{llama8B}）（Team，2024）、Phi-4（p_{phi4}）（14B；Abdin 等人，2024），以及一个使用人类基因组训练的 DNA 模型（p_{dna}）。[12] 对于第 7 节中的实验，我们使用 genlm-bytes[13] 将词元级模型转换为字节级模型。有关训练和评估设置的详情，请参见第 F 节；有关转换器的详情，请参见第 F.5 节。

回顾一下，转导语言模型 $p_{\mathcal{Y}}$ 由源模型 $p_{\mathcal{X}}$ 和转换器$\mathtt{f}$ 组成，后者编码了某个函数 f (§3)。为明确体现这种依赖关系，我们将 $p_{\mathcal{X}} \circ \mathtt{f}$ 定义为 $= p_{\mathcal{Y}}$，并将该操作称为复合。所有实验均使用效率更高的算法（§C.8）计算下一符号分布；该算法会在不同目标位置间复用缓存的分解，并利用结构性捷径（§C.5）。概率质量剪枝（§C.3）会保留概率最高的分解元素，使被丢弃的概率质量不超过剪枝阈值。

我们的实验涵盖了前缀单调性的不同情形（命题6.1）：词元到字节（T2B）是严格前缀单调的；DNA 到氨基酸是前缀单调的，并具有多符号前瞻；而 PTB 分词则不是前缀单调的。

从词元到字节。我们重新考察将模型从词元转换为字节的算法，如 Vieira 等人（2025a）所述。该变换可通过一个简单的转换器 $\mathtt{f}_\alpha$实现，其具有 $\mathcal{O}(|\mathcal{X}|)$个状态。具体而言，转换器为输入词汇表$\mathcal{X}$中的每个词元都包含一条链。这种结构使我们能够使用一种捷径（见图 14），仅通过一次语言模型调用就解析每个商元素。图 4 给出了这种机器的一个示例。我们在 wikitext-2-raw-v1 数据集（Merity 等人，2017 年）的前十个段落（对应前 7684 个字节）上，使用转换器 $p_{\text{gpt2}} \circ \mathtt{f}_\alpha$、 $p_{\text{llama1B}} \circ \mathtt{f}_\alpha$、$p_{\text{llama8B}} \circ \mathtt{f}_\alpha$和 $p_{\text{phi4}} \circ \mathtt{f}_\alpha$ 对第 5 节中的算法进行基准测试。如图 6（左）和 §G 中的表 8 所示，较低的剪枝阈值（τ）能使结果与参考分布（$\tau = 1\text{e-}5$）之间的 JSD 值更低，但会降低吞吐量；吞吐量以每秒字节数衡量。我们在 §G.1 的表 6 中确认，JSD 值与 Vieira 等人（2025a）取得的结果相近。他们的方法仅适用于严格前缀单调变换，因此能够使用一种专门的基于trie的算法，在保持准确度相近的同时实现更高的吞吐量。

从词元到正字法词边界。我们接下来考虑的变换，是将基于词元的语言模型转换为基于正字法单词的语言模型。此类单词的确切定义因应用场景而异。在某些场景中，像“wouldn’t”这样的缩约形式应视为一个整体。然而，在其他场景中，更自然的切分方式则是“would”和“n’t”。我们可以适配任何基于有限状态转换器的定义。语言学分词器（例如 PTB 分词器（Marcus 等人，1993））会利用上下文信息，将原始英语文本

[12]模型链接：https://huggingface.co/openai-community/gpt2-large，https://huggingface.co/meta-llama/Llama-3.2-1B，https://huggingface.co/meta-llama/Llama-3.1-8B，https://huggingface.co/microsoft/phi-4 和 https://huggingface.co/vesteinn/gpt2-dna。[13]genlm-bytes是 Vieira 等人(2025a）的实现；参见 https://github.com/genlm/genlm-bytes。

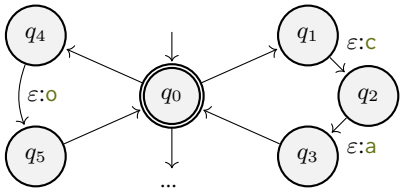


图 4：用于将词元模型转换为字符模型的有限状态转换器。以下是␣cat 和Dog。

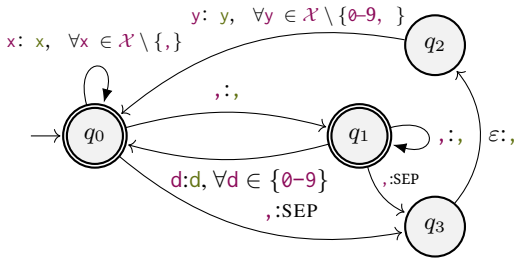


图 5：在逗号前插入分隔符（SEP）的有限状态转换器（FST）后接非数字字符。

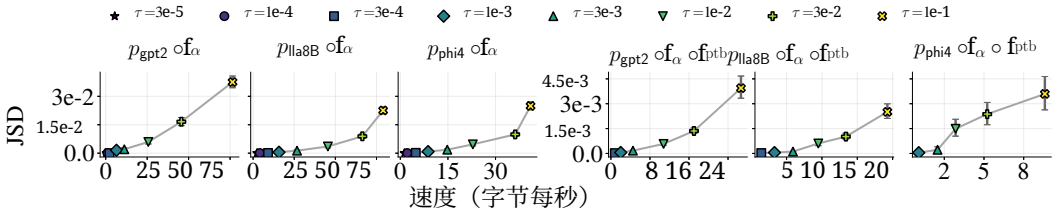


图 6：不同阈值下的平均詹森-香农散度（JSD）和吞吐量（字节/秒）τ。左：$p_{\mathcal{X}} \circ \mathbf{f}_\alpha$（参考值：$\tau$ = 1e-5）。右：$p_{\mathcal{X}} \circ \mathbf{f}_\alpha \circ \mathbf{f}^{\text{ptb}}$（参考值：$\tau$ = 1e-4；对于 p_{phi4}，为 3e-4）。柱状图显示 95% 的自助法置信区间。在最严格的阈值下，大规模的分解尺寸导致部分模型无法完成所有段落；参见 §G.2。

切分为适用于下游自然语言处理任务的、具有语言学意义的单元。我们构建了一个编码 PTB 分词器的有限状态转换器，$\mathbf{f}^{\text{ptb}}$；详见 §F.5。图 5展示了转换器中的一条规则：当逗号后面不是数字时，在逗号前插入 SEP。将$p_{\mathcal{X}} \circ \mathbf{f}_\alpha \circ \mathbf{f}^{\text{ptb}}$ 组合后，得到一个转导语言模型，将子词词元映射为 PTB 词元。为减少组合后有限状态转换器的状态数量并提高效率，我们使用来自 $p_{\mathcal{X}} \circ \mathbf{f}_\alpha$ 的字节转换模型来表示 `genlm-bytes`。[14] 在图 6（右）中，我们绘制了不同阈值 τ下的平均 JSD 与吞吐量的关系，使用的数据集与 §7中相同。参考分布使用τ = 1e-4（p_{phi4}时为 3e-4）。与 §7中的实验一样，我们观察到阈值越低，JSD 越低，但代价是吞吐量下降。使用较高阈值的实验见 §G，表 9。

将DNA模型转换为基于氨基酸的模型。 我们使用一个转换器，将由四种DNA核苷酸组成的序列转换为由二十二种氨基酸组成的序列。[15] 令 $p_{\text{dna}} \circ \mathbf{f}_{\text{dna2aa}}$表示将DNA核苷酸转换为氨基酸的转导模型。为评估我们的方法，我们对65种人类蛋白质进行采样。[16] 在表 10中，我们展示了不同阈值 τ下的平均JSD和吞吐量。请注意，由于分解后的预覆盖中候选项集合会随序列长度呈指数增长，该转换器尤其具有挑战性。为缓解组合爆炸，我们对候选集设置上限，并在改变该上限的同时报告吞吐量和JSD。

8 结论

我们提出了一个利用转换器转换语言模型的通用框架。通过实证结果，我们表明，我们的束求和近似能够高效地将基于词元的大语言模型转换为以字节、单词乃至氨基酸为单位的模型，而无需重新训练。我们的理论分析刻画了这类映射得以精确执行的条件。所提出的方法能够有效地重新利用现有语言模型，并准确计算由转换器定义的任意单位和转换的概率。通过将问题归结为转换器分解，该框架为利用有限状态方法在语言模型适配方面的未来进展铺平了道路。（未来工作与局限性详见 §J。）

[14]https://github.com/genlm/genlm-bytes，该项目实现了 Vieira 等人（2025a）提出的方法。[15]即，$\mathcal{X}$ = {A、C、G、T} 以及 $\mathcal{Y}$ = {A、R、N、D、C、Q、E、G、H、I、L、K、M、F、P、S、T、V、W、Y、B、Z、*}。[16] 从 https://www.uniprot.org/uniprotkb?query=Human 中采样，登录号见表 1。

致谢

作者感谢 Ben LeBrun、Brian DuSell、Clemente Pasti、Juan Luis Gastaldi、Mario Giulianelli 和 Yahya Emara 提供的宝贵反馈并参与讨论，这些意见和讨论极大地改进了本项工作。Vésteinn Snæbjarnarson 获得人工智能先锋中心的资助，资助来源为丹麦国家研究基金会，资助编号为 P1。

参考文献

Marah Abdin、Jyoti Aneja、Harkirat Behl、Sébastien Bubeck、Ronen Eldan、Suriya Gunasekar、Michael Harrison、Russell J. Hewett、Mojan Javaheripi、Piero Kauffmann、James R. Lee、Yin Tat Lee、Yuanzhi Li、Weishung Liu、Caio C. T. Mendes、Anh Nguyen、Eric Price、Gustavo de Rosa、Olli Saarikivi、Adil Salim、Shital Shah、Xin Wang、Rachel Ward、Yue Wu、Dingli Yu、Cyril Zhang 和 Yi Zhang。Phi-4 技术报告，2024。URLhttps://arxiv.org/abs/2412.08905。

Cyril Allauzen、Bill Byrne、Adrià de Gispert、Gonzalo Iglesias 和 Michael Riley。统计机器翻译中的下推自动机。 计算语言学，2014。URLhttps://aclanthology.org/J14-3008/。

Lisa Beinborn 和 Yuval Pinter。《分析子词分词的认知合理性》。载于自然语言处理实证方法会议论文集，2023。网址：https://aclanthology.org/2023.emnlp-main.272/。

Martin Berglund 和 Brink van der Merwe。《BPE 分词的形式化》。载于非经典自动机模型及应用国际研讨会论文集，第 388 卷，2023。网址：https://cgi.cse.unsw.edu.au/~eptcs/paper.cgi?NCMA2023.4.pdf。

Martin Berglund、Willeke Martens 和 Brink van der Merwe。《构造 BPE 分词确定有限自动机》。载于 自动机的实现与应用，2024。ISBN 978-3-031-71112-1。网址：https://link.springer.com/chapter/10.1007/978-3-031-71112-1_5。

Jean Berstel。转换与上下文无关语言，第38卷。Teubner，德国斯图加特，1979年。网址 https://www.worldcat.org/oclc/06364613。

Harry E. Blanchard、Alexander Pollatsek 和 Keith Rayner。阅读过程中旁中央凹词语信息的获取。《知觉与心理物理学》，46(1)，1989年。ISSN 0031-5117。网址 https://link.springer.com/content/pdf/10.3758/BF03208078.pdf。

Filippo Bonchi 和 Damien Pous。利用归纳法与余归纳破解非确定性。《*ACM*通讯》，2015年。网址https://doi.org/10.1145/2713167。

Christian Choffrut。作为有理关系的序列函数和子序列函数的刻画。《理论计算机科学》，5(3)，1977年。ISSN 0304-3975。网址https://doi.org/10.1016/0304-3975(77)90049-4。

Marco Cognetta、David Pohl、Junyoung Lee 和 Naoaki Okazaki。基于自动机的子词级约束生成中的陷阱、细微之处与技术。在分词研讨会，2025年。网址https://openreview.net/forum?id=DFybOGeGDS。

M. De Wulf、L. Doyen、T. A. Henzinger 和 J. F. Raskin。反链：一种用于检查有限自动机普遍性的新算法。载于国际计算机辅助验证会议论文集，2006年。网址https://doi.org/10.1007/11817963_5。

Edo Dotan、Gal Jaschek、Tal Pupko 和 Yonatan Belinkov。分词对生物序列 Transformers 的影响。《生物信息学》，40(4)，2024。ISSN 1367-4811。网址https://doi.org/10.1093/bioinformatics/btae196。

Philip Gage. 一种新的数据压缩算法。《*C* 用户杂志档案》，12，1994年。网址 https://api.semanticscholar.org/CorpusID:59804030。

Renato Geh、Honghua Zhang、Kareem Ahmed、Benjie Wang 和 Guy Van Den Broeck。分词空间中的信号在哪里？载于 自然语言处理实证方法会议论文集，2024年。网址：https://aclanthology.org/2024.emnlp-main.230/。

Marjan Ghazvininejad、Xing Shi、Yejin Choi 和 Kevin Knight。生成主题诗歌。载于自然语言处理实证方法会议论文集，2016年。网址：https://aclanthology.org/D16-1126/。Mario Giulianelli、Luca Malagutti、Juan Luis Gastaldi、Brian DuSell、Tim Vieira 和 Ryan Cotterell。论心理语言学中对分词的恰当处理。载于自然语言处理实证方法会议论文集，2024年。网址：https://aclanthology.org/2024.emnlp-main.1032。Kyle Gorman。Pynini：用于加权有限状态语法编译的 Python 库。载于统计自然语言处理与加权自动机 SIGFSM 研讨会论文集，2016年。网址：https:// aclanthology.org/W16-2409/。Jonathan Hayase、Alisa Liu、Noah A. Smith 和 Sewoong Oh。一次一个字节地从你的语言模型中采样。载于分词研讨会，2025年。网址：https://openreview.net/forum? id=DM8D9Nq9uO。John E. Hopcroft、Rajeev Motwani 和 Jeffrey D. Ullman。自动机理论、语言与计算导论。Addison-Wesley，第二版，2001年。ISBN 0201441241。网址：https://openlibrary.org/books/OL6795097M。Yanrong Ji、Zhihan Zhou、Han Liu 和 Ramana V Davuluri。DNABERT：用于基因组 DNA 语言的预训练双向变换器编码器表示模型。《生物信息学》，37(15)，2021年。ISSN 1367-4803。网址：https://doi.org/10.1093/bioinformatics/btab083。Arthur B. Kahn。大型网络的拓扑排序。《ACM 通讯》，5(11)，1962年。网址：https://dl.acm.org/doi/10.1145/368996.369025。Terry Koo、Frederick Liu 和 Luheng He。用于语言模型解码的基于自动机的约束。载于语言建模会议，2024年。网址：https://openreview.net/forum?id=BDBdblmyzY。Taku Kudo。子词正则化：使用多个子词候选项改进神经网络翻译模型。载于计算语言学协会年会论文集，2018年。网址：https://aclanthology.org/P18-1007/。Woosuk Kwon、Zhuohan Li、Siyuan Zhuang、Ying Sheng、Lianmin Zheng、Cody Hao Yu、Joseph Gonzalez、Hao Zhang 和 Ion Stoica。借助分页注意力实现大语言模型服务的高效内存管理。载于 SOSP，2023。网址：https://doi.org/10.1145/3600006.3613165。Alexander K. Lew、Tan Zhi-Xuan、Gabriel Grand 和 Vikash Mansinghka。使用概率程序对大语言模型进行序贯蒙特卡洛引导。载于 ICML 离散空间采样与优化研讨会，2023。网址：https://openreview.net/forum? id=Ul2K0qXxXy。Chu-Cheng Lin、Hao Zhu、Matthew R. Gormley 和 Jason Eisner。神经有限状态转换器：超越有理关系。载于计算语言学协会北美分会人类语言技术会议论文集，2019年。网址：https://aclanthology.org/N19-1024/。Mitchell P. Marcus、Beatrice Santorini 和 Mary Ann Marcinkiewicz。构建大型英语标注语料库：宾州树库。《计算语言学》，1993年。网址：https:// aclanthology.org/J93-2004/。Stephen Merity、Caiming Xiong、James Bradbury 和 Richard Socher。指针哨兵混合模型。载于国际学习表征会议，2017。网址：https: //openreview.net/forum?id=Byj72udxe。A. R. Meyer 和 L. J. Stockmeyer。带平方运算的正则表达式的等价性问题需要指数空间。载于开关与自动机理论年度研讨会论文集，1972年。网址：https://doi.org/10.1109/SWAT.1972.29。Mehryar Mohri。语言与语音处理中的有限状态转换器。《计算语言学》，1997。网址：https://aclanthology.org/J97-2003/。

Mehryar Mohri。《加权自动机的编辑距离：一般定义与算法》。《国际计算机科学基础杂志》，14(06)，2003。doi：10.1142/ S0129054103002114。网址 https://doi.org/10.1142/S0129054103002114。Mehryar Mohri。加权自动机算法，第5章。Springer Berlin Heidelberg，柏林，海德堡，2009。ISBN 978-3-642-01492-5。网址https://doi.org/10.1007/978-3-642-01492-5_6。Sathvik Nair 和 Philip Resnik。单词、子词和语素：在惊讶度与阅读时间的关系中，真正重要的是什么？载于计算语言学协会会议论文集：*EMNLP*，2023。网址 https://aclanthology.org/2023.findings-emnlp.752/。Eric Nguyen、Michael Poli、Marjan Faizi、Armin W Thomas、Michael Wornow、Callum Birch-Sykes、Stefano Massaroli、Aman Patel、Clayton M. Rabideau、Yoshua Bengio、Stefano Ermon、Christopher Re 和 Stephen Baccus。HyenaDNA：具有单核苷酸分辨率的长程基因组序列建模。载于神经信息处理系统大会论文集，2023。网址 https://openreview.net/forum?id=ubzNoJjOKj。Byung-Doh Oh 和 William Schuler。语言模型子词词汇表中的前导空格会混淆词概率的计算。载于自然语言处理经验方法会议论文集，2024。网址https://aclanthology.org/2024.emnlp-main.202/。J. Oncina、P. Garcia 和 E. Vidal。学习用于模式识别解释任务的后继转导器。《*IEEE*模式分析与机器智能汇刊》，15(5)，1993。网址https://ieeexplore.ieee.org/document/211465。OpenAI。Tiktoken：用于 OpenAI 模型的快速 BPE分词器，2024。网址 https://github.com/openai/tiktoken。乔治·奥威尔。《一九八四》。Secker & Warburg，1949。网址https://en.wikipedia.org/wiki/Nineteen_Eighty-Four。Buu Phan、Marton Havasi、Matthew Muckley 和 Karen Ullrich。《理解并缓解语言模型中的分词偏差》，2024。网址https://arxiv.org/abs/2406.16829。

Tiago Pimentel 和 Clara Meister。如何计算一个单词的概率。载于 自然语言处理实证方法会议论文集，2024 年。网址：https://aclanthology.org/2024.emnlp-main.1020/。

Jean-Éric Pin。 自动机理论的数学基础。LIAFA讲义，巴黎大学，2025年。链接 https://www.irif.fr/~jep/PDF/MPRI/MPRI.pdf。Jean-Éric Pin。 自动机理论手册。欧洲数学学会出版社，2021年。ISBN 978-3-98547-006-8。链接 https://doi.org/10.4171/Automata。乔立峰、叶鹏、任宇辰、白伟强、chaoqi liang、马新竹、董南青和欧阳万里。模型决定如何进行分词：使用 MxDNA 进行自适应 DNA 序列分词。发表于 神经信息处理系统年会，2024。链接https://openreview.net/forum?id=AQ1umQL7dZ。Michael O. Rabin 和 Dana Scott。有限自动机及其判定问题。 *IBM*研究与开发杂志，3(2)，1959年。链接 https://ieeexplore.ieee.org/document/5392601。Alec Radford、Jeff Wu、Rewon Child、David Luan、Dario Amodei 和 Ilya Sutskever。语言模型是无监督的多任务学习者。 *OpenAI*博客，2019年。链接 https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf。Pushpendre Rastogi、Ryan Cotterell 和 Jason Eisner。使用神经上下文为有限状态转换加权。发表于 计算语言学协会北美分会人类语言技术会议论文集，2016年。doi：10.18653/v1/N16-1076。链接 https://aclanthology.org/N16-1076/。

Keith Rayner、Arnold D. Well、Alexander Pollatsek 和 James H. Bertera。阅读时注视点右侧有用信息的可用性。《知觉与心理物理学》，31(6)，1982。链接https://doi.org/10.3758/BF03204186。

Rico Sennrich、Barry Haddow 和 Alexandra Birch。《使用子词单元对罕见词进行神经机器翻译》。载于 计算语言学协会年会论文集，2016年。网址：https://aclanthology.org/P16-1162/。

Xinying Song、Alex Salcianu、Yang Song、Dave Dopson 和 Denny Zhou。《快速 WordPiece 词元化》。载于 自然语言处理实证方法会议论文集，2021年。网址：https://aclanthology.org/2021.emnlp-main.160/。

Felix Stahlberg、Christopher Bryant 和 Bill Byrne。《使用有限状态转换器进行神经语法错误纠正》。载于 计算语言学协会北美分会人类语言技术会议论文集，2019年。网址：https://aclanthology.org/N19-1406/。

Llama 团队。《LLaMA 3 模型家族》，2024。网址：https://arxiv.org/abs/2407.21783。

Tim Vieira、Benjamin LeBrun、Mario Giulianelli、Juan Luis Gastaldi、Brian DuSell、John Terilla、Timothy J. O'Donnell 和 Ryan Cotterell。从基于词元的语言模型到基于字符的语言模型。载于 第四十二届国际机器学习会议，2025a。网址：https://openreview.net/forum?id=sQS0roNQZR。

Tim Vieira、Tianyu Liu、Clemente Pasti、Yahya Emara、Brian DuSell、Benjamin LeBrun、Mario Giulianelli、Juan Luis Gastaldi、John Terilla、Timothy J. O'Donnell 和 Ryan Cotterell。基于规范字节对编码的语言模型。载于 国际机器学习会议，2025b。网址：https://openreview.net/forum?id=eCVrfVDNSY。

Ning Wang、Jiang Bian、Yuchen Li、Xuhong Li、Shahid Mumtaz、Linghe Kong 和 Haoyi Xiong。通过基序感知预训练和类型引导微调进行多用途 RNA 语言建模。《自然·机器智能》，6(5)，2024。网址：https://doi.org/10.1038/s42256-024-00836-4。

Xuezhi Wang、Jason Wei、Dale Schuurmans、Quoc V Le、Ed H. Chi、Sharan Narang、Aakanksha Chowdhery 和 Denny Zhou。《自洽性提升语言模型中的思维链推理》。载于 国际学习表征会议，2023年。网址：https://openreview.net/forum?id=1PL1NIMMrw。

Brandon T. Willard 和 Rémi Louf。面向大语言模型的高效引导生成，2023。网址：https://arxiv.org/abs/2307.09702。

Thomas Wolf、Lysandre Debut、Victor Sanh、Julien Chaumond、Clement Delangue、Anthony Moi、Pierric Cistac、Tim Rault、Remi Louf、Morgan Funtowicz、Joe Davison、Sam Shleifer、Patrick von Platen、Clara Ma、Yacine Jernite、Julien Plu、Canwen Xu、Teven Le Scao、Sylvain Gugger、Mariama Drame、Quentin Lhoest 和 Alexander Rush。《Transformers：最先进的自然语言处理》。载于 自然语言处理实证方法会议论文集：系统演示，2020年。网址：https://aclanthology.org/2020.emnlp-demos.6。

Yonghui Wu、Mike Schuster、Zhifeng Chen、Quoc V. Le、Mohammad Norouzi、Wolfgang Macherey、Maxim Krikun、Yuan Cao、Qin Gao、Klaus Macherey、Jeff Klingner、Apurva Shah、MelvinJohnson、Xiaobing Liu、Łukasz Kaiser、Stephan Gouws、Yoshikiyo Kato、Taku Kudo、Hideto Kazawa、Keith Stevens、George Kurian、Nishant Patil、Wei Wang、Cliff Young、Jason Smith、Jason Riesa、Alex Rudnick、Oriol Vinyals、Greg Corrado、Macduff Hughes 和 Jeffrey Dean。《谷歌神经机器翻译系统：弥合人类与机器翻译之间的差距》，2016年。网址：https://arxiv.org/abs/1609.08144。

Vicky Xefteri、Tim Vieira、Ryan Cotterell 和 Afra Amini。《通过后验推断对语言模型进行句法控制》。载于 Wanxiang Che、Joyce Nabende、Ekaterina Shutova 和 Mohammad Taher Pilehvar（编），计算语言学协会论文集：*ACL 2025* 研究成果，第 25350–25365 页，奥地利维也纳，2025年7月。计算语言学协会。ISBN 979-8-89176-256-5。doi：10.18653/v1/2025.findings-acl.1300。网址：https://aclanthology.org/2025.findings-acl.1300/。

附录目录

符号表 17

B 关于转换器的背景 1 **8**

C 高效算法 1 **9**

C.1 目标：自回归接口的高效实现 . . . . . 19 C.2 基于前沿的检查（is_柱集、is_成员、is_活性）.21 C.3 剪枝（prune）.22 C.4 快速下一符号分解（decompose_next）.23 C.5 下一符号分解捷径.25 C.6 输入投影普适性快捷方法（fast_univ_filter）.26 C.7 记忆化与批处理 of $p_{\mathcal{X}}$ .26 C.8 实现自回归接口.27 C.9 全通用快速路径.28

D 有限性条件的证明30

D.1 前缀单调映射的性质.30D.2 严格前缀单调映射的商集界.31D.3 有限商与余项的充分条件.32

E 分解示例34

F 实验设置3 **6**

F.1 数据集.36F.2 模型.36F.3 参数.36F.4 GPU 使用.36F.5 实验中使用的转换器详情.37

G 补充实验结果 39

G.1 与 Vieira 等人（2025a）的比较 .39G.2 詹森–香农散度 .40G.3 分解大小 .43G.4 交叉熵.45G.5 计算捷径基准测试 .46

H 基于状态的分解 **47**

I 相关工作 **48**

J 局限性 **50**

A 符号表

符号	释义
ε	空字符串
EOS	字符串结束符号
$x, x' \in \mathcal{X}$	源字母表中的符号 $\mathcal{X}$
$\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}^*$	源字符串
$\mathcal{X}^*$	所有源字符串的集合
$Z, Z' \subseteq \mathcal{X}^*$	源字符串集合
$\boldsymbol{x}\boldsymbol{x}'$	拼接（字符串）
ZZ'	拼接（字符串集合）
$y, y' \in \mathcal{Y}$	目标字母表中的符号
$\boldsymbol{y}, \boldsymbol{y}' \in \mathcal{Y}^*$	目标字符串
$\mathcal{Y}^*$	所有目标字符串的集合
$f: \mathcal{X}^* \to \mathcal{Y}^*$	从源字符串到 $\mathcal{X}^*$ 目标字符串的转换 $\mathcal{Y}^*$
$\boldsymbol{x} \preceq \boldsymbol{x}'$	$\boldsymbol{x}$是 $\boldsymbol{x}'$的前缀
$\boldsymbol{x} \prec \boldsymbol{x}'$	$\boldsymbol{x}$ 是 $\boldsymbol{x}'$的严格前缀
$\langle Z \rangle$	由 Z张成的柱集
$\mathrm{pf}(Z)$	Z 的前缀基
$p_{\mathcal{X}}$	针对源字符串 $\mathcal{X}^*$ 的语言模型
X	$\mathcal{X}^*$ 值随机变量 $X \sim p_{\mathcal{X}}$
$\overrightarrow{p_{\mathcal{X}}}$	$p_{\mathcal{X}}$ 的前缀概率
$p_{\mathcal{Y}}$	针对目标字符串 $\mathcal{Y}^*$的语言模型
$f(X)$	$\mathcal{Y}^*$值随机变量 $f(X) \sim p_{\mathcal{Y}}$
$\overrightarrow{p_{\mathcal{Y}}}$	$p_{\mathcal{Y}}$的前缀概率
f	f的转换器实现
$\hookrightarrow \mathcal{X}$	输入字母表
$\hookrightarrow \mathcal{Y}$	输出字母表
$\hookrightarrow \mathsf{S}$	状态集合
$\hookrightarrow \mathsf{I} \subseteq \mathsf{S}$	初始状态集合
$\hookrightarrow \mathsf{F} \subseteq \mathsf{S}$	接受状态集合
$\hookrightarrow \mathsf{T}$	转移集合
$\hookrightarrow \mathsf{U} \subseteq \mathsf{S}$	IP-全称状态集合（§C.6）
$s, s' \in \mathsf{S}$	状态
$(s \xrightarrow{x:y} s') \in \mathsf{T}$	从 s 转移到 s'，扫描 x 并发出 y
$\mathsf{T}(s) \subseteq \mathsf{T}$	从状态 s出发的出边
$\mathsf{f} \circ \mathsf{f}'$	转换器组合
$f \circ f'$	关系组合
投射$_x(\mathsf{f})$	输入投影
$\mathsf{f}_{[s]}$	强制开始 f 在状态 s
$\boldsymbol{y}\mathcal{Y}^*$	接受 $\boldsymbol{y}\mathcal{Y}^*$ 的复制转换器，或对应的关系。
$\mathsf{f} \circ \boldsymbol{y}\mathcal{Y}^*$	其路径仅限于接受 $\boldsymbol{y}\mathcal{Y}^*$的转换器。
$f^{-1}(\boldsymbol{y})$	目标字符串的原像 $\boldsymbol{y}$ (§3)
$f^{-1}(\mathcal{Y})$	集合的原像 $\mathcal{Y}$, $\{f^{-1}(\boldsymbol{y}) \mid \boldsymbol{y} \in \mathcal{Y}\}$ (§3)
$\mathcal{P}(\boldsymbol{y})$	目标字符串的预覆盖 $\boldsymbol{y}$ (§3)
$\mathcal{C}(\boldsymbol{y})$	包含于其中的最大柱集 $\mathcal{P}(\boldsymbol{y})$ (§4)
$\mathcal{Q}(\boldsymbol{y})$	商 (§4)
$\mathcal{R}(\boldsymbol{y})$	余项（§4）

记号约定。颜色表示所属领域：洋红色表示源领域对象（例如，$\boldsymbol{x}$, $\mathcal{X}$,$\mathcal{P}$, $\mathcal{Q}$, $\mathcal{R}$），橄榄绿表示目标领域对象（例如，$\boldsymbol{y}$, $\mathcal{Y}$, f）。字体表示类型：符号使用小写斜体（x, y），字符串使用粗斜体（$\boldsymbol{x}$, $\boldsymbol{y}$），字母表和集合使用花体（$\mathcal{X}$, $\mathcal{Y}$, $\mathcal{P}$,$\mathcal{Q}$, $\mathcal{R}$），转导器组件使用无衬线字体（S, T, I, F, U），转换器本身（f）以及示例中的具体字符串（abc, xyz）使用打字机字体。上方带箭头的 $\rightarrow$标记前缀概率：$\overrightarrow{p_{\mathcal{X}}}$是 $p_{\mathcal{X}}$的前缀概率。

B 转换器背景

本节介绍转换器的补充信息，以补充 §2。

转换器的变体。如果转换器定义了一个函数，我们称它为函数式；如果它定义了一个偏函数，则称它为部分函数式。如果对于每个状态 $s \in \mathrm{S}$，$|\mathrm{T}(s,\varepsilon)| = 0$ 和 $|\mathrm{T}(s,x)| \leq 1$ 对所有 $x \in \mathcal{X}$[17]都成立，则称转换器是输入确定性的。对于输入确定性转换器，每个源字符串 $\boldsymbol{x} \in \mathcal{X}^*$ 至多有一条扫描 $\boldsymbol{x}$ 的接受路径，因此至多能输出一个目标字符串 $\boldsymbol{y} \in \mathcal{Y}^*$。因此，每个输入确定性转换器都定义了一个（偏）函数。复制转换器是指每次转移输出的符号都与其扫描的符号相同的转换器，即每次转移的形式都是 $s \xrightarrow{x:x} s'$。我们将复制转移简记为 $s \xrightarrow{x} s'$.[18] 复制转换器定义偏恒等函数：它们将指定子集中的每个字符串映射为其自身，并丢弃其余字符串。正如我们简记复制转移一样，也可以通过 $(\boldsymbol{x},\boldsymbol{x}) \mapsto \boldsymbol{x}$ 将它们隐式映射到一个字符串集合。

操作。转换器支持复合：给定转换器 f 和 f′，其复合 f ∘ f′是表示 $[\![\mathrm{f} \circ \mathrm{f}']\!] \stackrel{\text{def}}{=} [\![\mathrm{f}]\!] \circ [\![\mathrm{f}']\!]$。[19][20] 我们用 $\mathrm{f} \circ \boldsymbol{y}\mathcal{Y}^*$表示编码关系$[\![\mathrm{f}]\!] \circ \{(\boldsymbol{y}', \boldsymbol{y}') \mid \boldsymbol{y}' \in \langle \boldsymbol{y} \rangle\}$ 的转换器。只要上下文意图清晰，我们会在这些表示之间自由转换。我们定义输入投影操作，用于编码关系$[\![\mathrm{proj}_{\mathcal{X}}(\mathrm{f})]\!] = \{(\boldsymbol{x},\boldsymbol{x}) \mid \exists \boldsymbol{y}\colon (\boldsymbol{x},\boldsymbol{y}) \in [\![\mathrm{f}]\!]\}$，其中 $\mathrm{proj}_{\mathcal{X}}(\mathrm{f}) \stackrel{\text{def}}{=} (\mathrm{S}\ \mathcal{X}, \mathcal{X}, \mathrm{I}, \mathrm{F}\ \{(s\ \xrightarrow{x:x}\ s') \mid s\ \xrightarrow{x:y} s' \in \mathrm{T}\})$，这是一个复制转换器。令 $\mathrm{f}_{[s]}$ 表示在状态 s中强制起始 f 的操作，$\mathrm{f}_{[s]} \stackrel{\text{def}}{=} (\mathrm{S}\ \mathcal{X}, \mathcal{Y}, \{s\}, \mathrm{F}, \mathrm{T})$；该操作生成一台机器，用于定义由从给定 s 开始并在接受状态结束的路径所生成的源—目标后缀对集合。如果一个状态 s 满足 $[\![\mathrm{proj}_{\mathcal{X}}(\mathrm{f}_{[s]})]\!] = \mathcal{X}^*$，我们就称其为 IP-通用（输入投影全覆盖），也就是说，无论后续输入是什么，该转换器仍能产生输出。令 $\mathrm{U} \subseteq \mathrm{S}$ 表示 IP-通用状态集合；对于每个转换器，都可以预先计算出该集合（§C.6）。

我们的算法使用一种**输入确定性化**转换$\mathsf{determinize}(\mathtt{f})$，将一个（部分）函数式转换器$\mathtt{f}$ 映射为与其等价且具有输入确定性的转换器。一般而言，这种映射并不总能通过有限数量的状态实现。[21] 然而，对于复制转换器这一特殊情况，输入确定性化始终是可能的，但在最坏情况下可能导致指数级膨胀。[22] 我们使用$\mathsf{trim}(\mathtt{f})$ 表示一种**裁剪**操作，该操作会移除所有不出现在任何接受路径上的状态和边。这些都是任何有限状态转换器库中都已实现的标准操作；更多细节可参见（Pin，2021年；2025年）。

图示符号。我们使用§E 中所示的图来表示转换器。不带源状态的转移表示初始状态；双线状态表示接受状态。转移$s\ \xrightarrow{x:y}\ s'$ 以状态之间的箭头表示。

局限性。有限状态转换器定义了有理关系（Berstel，1979，第 III 章）这一类关系。由于有限状态转换器只有有限个状态，因此它们能够表示的关系本质上受到限制。例如，有限状态转换器无法执行需要无界匹配或计数的变换。相比之下，具有无界内存的转换器超越了有理类，具有更强的表达能力，但也带来了更高的复杂性，并且关键性质（例如普遍性）通常是不可判定的。

[17]子序列转换器这一更一般的概念还包括一个最终输出函数；对于复制转换器而言，不存在这一差异。[18]对于熟悉有限状态自动机的读者来说，每个复制转换器都同构于一个接受器。[19]在这里，关系组合定义为 $f \circ g \stackrel{\text{def}}{=} \{(x,z) \mid (x,y) \in f,\ (y,z) \in g\}$，其中 f和 g 是关系。[20]Pin（2025年，第XIX章，第2节）给出了一种构造 f ∘ f′的高效方法。Choffrut（1977年）通过在输入侧使用幂集构造（如确定化非确定有限自动机时所用的构造）给出了这样一种算法。他证明，当且仅当自动机具有有界变差时，该方法才会产生有限机器；有界变差是指：对于任意两个前缀距离（即移除两字符串最长公共前缀后所得字符串长度之和）有界的字符串，其输出前缀距离也有界。他还给出了一个可测试的确定化条件，称为孪生条件：一旦两个 run 读取了相同的输入前缀（即我们在相同位置切分输入），那么对于每个共同的延续，它们追加的后续输出也相同。[22]对于熟悉有限状态自动机的读者来说，复制转换器的输入确定性化与等价有限状态自动机的确定化同构，见§B。

C 高效算法

第 §5 节的分解算法作用于经过确定化和修剪的预覆盖机P =trim（determinize（proj$\mathcal{X}$（f $\circ\, y\mathcal{Y}^*$））。实际上，与 $y\mathcal{Y}^*$ 进行复合以及显式确定化的代价都很高，并且必须针对每个目标 y重复执行。本节中的算法以惰性方式执行这两个步骤，直接作用于原始转换器f ，避免显式实例化。这要求通过前沿（§C.1.1）仔细记录部分输出，但能够预先计算 IP-通用状态，并在不同目标之间增量复用。

我们首先陈述算法目标（§C.1），然后介绍前沿数据结构（§C.1.1），并展示如何将每项检查归结为一次前沿查询（§C.2）。我们在全文中将 (S $\mathcal{X}, \mathcal{Y}$, I, F, T) = f 记为 $\overrightarrow{p_{\mathcal{X}}}$，因为上下文已足以说明，因此省略对它的依赖。

C.1 目标：高效实现自回归接口

我们旨在为转换语言模型的前缀概率 $\overrightarrow{p_{\mathcal{Y}}}(y)$、字符串概率 $p_{\mathcal{Y}}(y)$以及下一个词元分布 $\overrightarrow{p_{\mathcal{Y}}}(\cdot \mid y)$设计高效算法。具体而言，我们希望高效实现以下方法，它们构成了转换语言模型的接口（§2）。

44 前缀_概率(y):45 (Q, R) $\leftarrow$ 分解(y)46 返回 $\sum_{x\in Q} \overrightarrow{p_{\mathcal{X}}}(x) + \sum_{x\in R} p_{\mathcal{X}}(x)$47 概率($y$):48 返回 前缀_概率($y$) ·下一个_分布($y$)[EOS]

49 **定义** next_dist(y): 50 $p \leftarrow \{\}$ 51 $Z \leftarrow$ 前缀_概率(y)52 **对于** $y' \in \mathcal{Y}$: 53 $p[y'] \leftarrow$ 前缀_概率(y)54 $p[\ ] \leftarrow 1 - \sum_{y'\in\mathcal{Y}} p[y']$ 55 **返回** p $y y' / Z$ 概率()EOS

上述基本操作是前缀_概率，它需要调用一次分解。next_dist 和 prob 都由它派生而来：next_dist 计算 $|\mathcal{Y}|$ 个额外的前缀概率，并通过补集得到 $\overrightarrow{p_{\mathcal{Y}}}$（EOS $\mid y$）；prob 是一个单行乘积。在实现中，应对 $\overrightarrow{p_{\mathcal{X}}}$和 $p_{\mathcal{X}}$进行记忆化，这样将 $\overrightarrow{p_{\mathcal{X}}}$（$x$）扩展到 $\overrightarrow{p_{\mathcal{X}}}$（$xx$）时，只需进行一次条件评估，而不必重放全部历史记录（参见 §C.7）。在 §C.4 中，我们展示了如何通过更高效的联合分解来计算 next_dist，而不是 $|\mathcal{Y}|+1$ 分别调用前缀_概率。

图 2 中的三项检查——是否为_柱集、是否为_成员，以及是否_活性——都需要确定读取源字符串 x 后哪些转换器状态可达，以及每个状态相对于 y 已生成了什么输出。我们用一个数据结构来记录这些信息：即前沿 (§C.1.1)。在 §C.2 中，我们将说明如何将每项检查归结为对前沿的一次简单查询。实际上，商与余项可能很大，甚至是无限的，因此我们引入剪枝（§C.3）来获得实用的近似。

C.1.1 前沿计算

即使 f 是函数式的，它也可能具有非确定性，因此对于给定的源字符串 x 和目标 y，转换器 f 可能同时到达多个状态。这些路径中的每一条都可能产生了不同的输出缓冲区。前沿 $\mathcal{F} = \text{run}_y(x)$ 收集了这些信息：它是可从初始状态读取 x 后到达的 (s, b) 对的集合——其中 $s \in$ S 是转换器状态，$b \in \mathcal{Y}^*$ 是目前为止生成的输出——并且只保留与 y兼容的缓冲区 b 。[23] 前沿编码了进行三项检查所需的全部信息：是否为_柱集、是否为_成员，以及是否_活性，都可以仅根据 $\mathcal{F}$ 进行判断，无需保留完整的路径历史（§C.2；图 8）。图 7 中的伪代码描述了如何使用三个函数 run$_y$、step$_y$ 和 closure$_y$计算前沿。

前沿取代了第§5节中使用的显式预覆盖机 P = 裁剪(确定化(投影$_{\mathcal{X}}$(f $\circ\, y\mathcal{Y}^*$))) 。前沿并不将 P 实体化，而是直接在f上跟踪相同的信息，从而避免了与 $y\mathcal{Y}^*$ 的复合以及显式确定化。该设计体现了两阶段结构：在路径的缓冲区覆盖目标之前，前沿跟踪按目标兼容性筛选的每个状态的输出缓冲区 (s, b)；一旦缓冲区达到或超过 y，便可截短前沿以进行全称性检查（§C.2）。前沿将这两个阶段统一为一个惰性数据结构。

[23]缓冲区 b 与.....兼容 y 当且仅当 $b \preceq y \vee y \preceq b$，也就是说，缓冲区要么已经覆盖目标，要么尚未偏离目标。

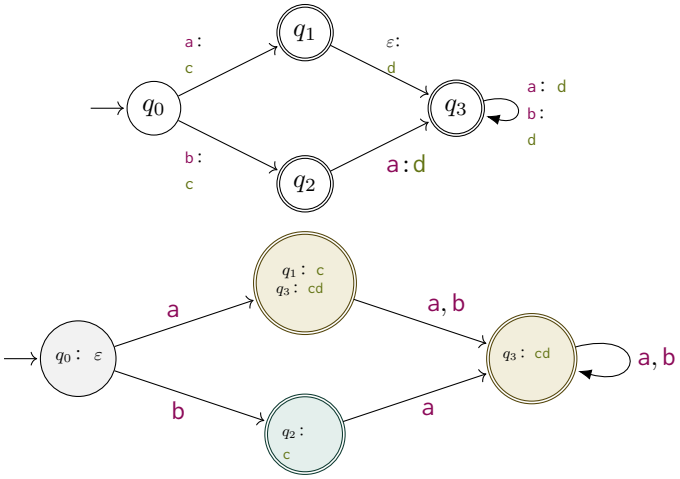
```
def run_y(x_1 ⋯ x_N):   # 记忆化
    如果 N = 0: 返回 闭包_y(I × {ε})
    返回 步数_y(run_y(x_{<N}) x_N)

定义 步数_y(F, x):
    F' ← ∅
    遍历 (s, b) 在 F:
        对 (_ -x:y-> s') ∈ T(s, x):
        b' ← by
            如果 b' ⪯ y ∨ y ⪯ b':
                F'.add((s', b'))
    返回 闭包_y(F')
```

```
定义 闭包_y(F):
    F' ← F
    q ← QUEUE(F)
    当 |q| > 0:
        (s, b) ← q。弹出()
        对于 (_ -ε:y'-> s') ∈ T(s, ε) : b' ← by'

            如果 b' ⪯ y ∨ y ⪯ b':
                如果 (s', b') ∉ F':
                    F'。添加((s', b'))
                    q。添加((s', b'))
    返回 F'
```

图 7：基于前沿的状态机



x	前沿	为_活性	是_柱集	是_成员	类
ε	$\{(q_0,\ \varepsilon)\}$	✓	✗	✗	扩展
a	$\{(q_1,\ \mathsf{c}),\ (q_3,\ \mathsf{cd})\}$	✓	✓	—	$\mathcal{Q}(\mathsf{c})$
b	$\{(q_2,\ \mathsf{c})\}$	✓	✗ (q_2 在b上失效	✓	$\mathcal{R}(\mathsf{c})$

图 8：目标字符串 $y = \mathsf{c}$ 的截断前沿示例。上：转换器 f。中：前沿图 $\mathsf{run}_{\mathsf{c}}$，其节点是由 (s, b) 对组成的集合，边是输入符号。缓冲区被截断为长度 $|y|+1 = 2$：对 (q_3, cd) 读入任意输入都会产生 cdd，再截断回 cd，从而形成自环。截断在保持缓冲区是否已达到目标以及其承诺的下一个输出符号不变的同时，使状态空间保持有限。读取 $x = \mathsf{a}$ 后，通过 ε-闭包到达两个状态：q_1 (缓冲区 $= y$) 和 q_3 (缓冲区 $\succ y$)。黄色节点是全称的——每个输入都会导向非空的接受后继——因此 a 属于 $\mathcal{Q}(\mathsf{c})$。青绿色节点 q_2 在 b 上没有弧，因此广度优先搜索失败，b 进入 $\mathcal{R}(\mathsf{c})$。

前沿具有一个有用的单调性：将目标扩展一个符号只会从前沿中移除元素，而不会添加新元素。这意味着我们可以增量计算前沿——随着目标增长进行筛选，而不是重新构建。

命题 C.1 (前沿包含)。 对于任意源字符串 x、目标字符串 $y_1 \cdots y_N$ 满足 $N > 0$：

$$\mathsf{run}_{y_1 \cdots y_N}(x) = \{(s, b) \in \mathsf{run}_{y_1 \cdots y_{N-1}}(x) : |b| < N \vee b_N = y_N\} \tag{9}$$

证明。 对 步数 和 闭包 进行缓冲区过滤，只保留 (s, b) 这类配对，其中 b 与目标在它们共有的前缀上相一致。由于有限状态转换器弧只会向缓冲区追加内容，一旦 b 足够长，包含一个 N^{th}符号，而 $b_N \neq y_N$，那么所有后代缓冲区都会被 $y_1 \cdots y_N$兼容性过滤器过滤掉。因此，在末尾对 $\mathsf{run}_y(x)$ 进行过滤，等同于在每一步进行过滤。

要理解这一点，请注意，($\supseteq$) 方向成立，因为 $y_1 \cdots y_N$相容元素也满足$y_1 \cdots y_{N-1}$兼容性（前者的限制更严格）。($\subseteq$) 方向成立，因为那些满足 $y_1 \cdots y_{N-1}$兼容性但 不满足 $y_1 \cdots y_N$兼容性的元素（即 $|b| \geq N$和 $b_N \neq y_N$）无法产生后来变得满足 $y_1 \cdots y_N$兼容性的后代，因为缓冲区只会增长。■

命题 C.2. 命题 *C.1* 推广到单符号扩展之外：对于任意 $y' \preceq y$:

$$\mathsf{run}_y(x) = \{(s,b) \in \mathsf{run}_{y'}(x) \colon b \preceq y \vee b \succeq y\} \tag{10}$$

证明。对 $|y| - |y'|$使用归纳法。基础情形 $y = y'$ 显然成立。对于归纳步骤，写作$y = y'y_N$，其中 $y' \preceq y'$。根据命题 C.1，$\mathrm{run}_y(x) = \{(s,b) \in \mathrm{run}_{y'}(x) \colon |b| < N \vee b_N = y_N\}$。根据归纳假设，$\mathrm{run}_{y'}(x) = \{(s,b) \in \mathrm{run}_{y'}(x) \colon b \preceq y' \vee b \succeq y'\}$。代入后，$(s,b) \in \mathrm{run}_{y'}(x)$ 能通过两个过滤器，当且仅当 b 在截至 $\min(|b|, N)$ 的所有位置上都与 y 一致，即$b \preceq y \vee b \succeq y$。■

命题 C.2 是实现从任意目标前缀的缓存前沿热启动 分解_的下一步 (§C.4) 的关键性质。

C.2 基于前沿的 检查 (是否为_柱集，是否为_成员，是否_活性)

回顾一下，分解算法（图 2）依赖于三项检查（是否为_柱集、是否为_成员、是否_活性）。本节介绍如何根据前沿计算这些检查。

79 定义 is_cylinder(x, y): # 对 run$_y(x)$ 的截断缓冲区 82 $V \leftarrow \{\mathcal{F}\}$ **进行记忆化；队列 q** $\leftarrow$**QUEUE($\{\mathcal{F}\}$) 83 当 q 非空时：** 84 $\mathcal{F} \leftarrow$**q.pop() 85 # 接受条件：某一对匹配 86 # 目标且为接受状态 87 如果** $\neg\exists(s,b) \in \mathcal{F}$**:** $b \succeq y \wedge s \in$**F: 88 返回 False 96 定义 is_member(x, y): 97 返回** $\exists(s,b) \in$**run$_y(x)$:** 98 $b \succeq y \wedge s \in$**F 99 定义 is_live(x, y):** 100 $\mathcal{F}$ **run ( ) 定义截断缓冲区 _buf$_y$( ):** 103 $N \leftarrow |y|$ 102 **返回** $\{(s, b_{\leq N}) \mid (s,b) \in \mathcal{F},$ 101

104 $b \preceq y \vee y \preceq b\}$ **89 # 完备性：每个输入都会导向一个 90 # 非空后继 91 对于** $x' \in \mathcal{X}$**:** 92 $\mathcal{F}' \leftarrow$ **trunc_buf$_y$(step$_y(\mathcal{F}, x')$) 93 如果** $\mathcal{F}' = \emptyset$**: 返回 False 94 如果** $\mathcal{F}' \notin V$**:** V**.add($\mathcal{F}'$); q.push($\mathcal{F}'$) 95 返回 True**

图 9：基于前沿的检查。is_柱集(x, y) 通过对投影前沿执行幂集广度优先搜索来验证普遍性。is_成员检查是否存在接受覆盖状态；is_活性检查前沿是否非空。

要检查 $\langle x \rangle \subseteq \mathcal{P}(y)$，基于前沿的 is_柱面会对前沿 run$_y(x)$ 应用截断_缓冲区操作，将缓冲区截断至长度 $|y|$，从而得到一个截断前沿——由缓冲区与 y 兼容的 (s,b) 对组成的集合。随后，它通过在截断前沿上进行广度优先搜索来验证普遍性：每个后继前沿都通过先应用 step$_y$ 、再应用截断_缓冲区操作得到。与缓冲区可以无限增长的普通前沿不同，截断缓冲区必须是 y 的前缀，因此最多只有$|y|+1$ 种不同的缓冲区，使得截断前沿的状态空间有限。每个截断前沿都对应于确定化预覆盖机 P 的一个状态，因此广度优先搜索会终止。广度优先搜索会检查每个截断前沿是否既接受（某个对具有 $b \succeq y$ 和 $s \in$ F），又完整（每个输入符号都会导向一个非空后继）。如果不进行这种遍历，两条非确定性路径可能产生相同的字符串，却分别以不同状态 s_1 和 s_2结束；这两个状态单独来看都不具有普遍性，尽管 $[\![\mathbf{f}_{[s_1]}]\!] \cup [\![\mathbf{f}_{[s_2]}]\!] = \mathcal{X}^*$.[24] 。IP-通用状态的预计算和柱面检查捷径详见 §C.6。

[24]有限状态自动机的普遍性是可判定的；高效算法包括基于反链的模拟（De Wulf等人，2006年）、至同余的双模拟（Bonchi与Pous，2015年）以及经典等价性检查（Meyer与Stockmeyer，1972年）。

当柱面检查失败时，成员资格问题可归结为检查前沿是否包含一个覆盖接受对（$\boldsymbol{b} \succeq \boldsymbol{y}$ 和 $s \in \mathsf{F}$）：如果包含，$\boldsymbol{x} \in \mathcal{P}(\boldsymbol{y})$ 和 $\boldsymbol{x}$ 进入 $\mathcal{R}(\boldsymbol{y})$。
活性是指前沿非空。由于前沿中的配对已筛选为目标兼容缓冲区，因此前沿非空意味着某个扩展 $\boldsymbol{x}$ 可以到达预覆盖。

C.3 剪枝(剪枝)

即使分解是有限的，商与余项也可能非常庞大。枚举所有这些内容通常并不实际，因此剪枝对于可扩展性至关重要。[25] 至少，剪枝会丢弃前缀概率为零的源字符串，因为它们无法贡献任何质量：

```
def prune(q′):
    return {x ∈ q′ : p⃗_X(x) > 0}
```

这个简单的筛选器只会丢弃概率为零的字符串。在实践中，我们采用概率质量剪枝策略：给定总质量为 $Z = \sum_{\boldsymbol{x} \in \mathsf{q}'} \overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})$ 的候选项 q′，我们按 $\overrightarrow{p_{\mathcal{X}}}$ 降序排序，并贪心地保留候选项，直到其累积质量达到 $(1-\tau) \cdot Z$ 或达到容量 n_{max}，以先发生者为准。由此得到的分解是近似的：每步被丢弃的候选项最多贡献 $\tau \cdot Z$ 质量，但随着 $\tau \to 0$ 增大，误差会趋于消失。

```
107 def 剪枝_概率_质量(τ, n_max): 108 def 剪枝(q′):
109     M ← | ′| q q sort_降序(q′, key=p⃗_X)
110     ′ ←
111             x        x ∈ ′]      w_1 ≥ ··· ≥ _M
112     ←
w ← [p⃗_X( ) for q # w W 累积_求和(w)
113     Z ← W_114          ← ∅_M q 115 for
 m ∈ {1, ... , min(n_max, M)}: 116 q.add(q′_m) 117 if
W_m ≥ (1 − τ)Z: break 118 return q 119 return
剪枝
```

近似界。 在每一步，被丢弃的质量至多为 $\tau \cdot Z$。随着 $\tau \to 0$ 和 $n_{\mathsf{max}} \to \infty$，该算法收敛到精确分解。

有限终止。 即使分解是有限的，其规模也可能极为庞大。剪枝通过将每步的队列限制为 n_{max} 个候选项，使算法变得可行，并且还保证分解为无限的转换器的终止性（§D.3）。

回溯。剪枝偶尔可能导致死胡同，即不再存在有效的扩展。在对观测序列进行评分时，如果发生这种情况，下一符号 y 在近似 $\overrightarrow{p_{\mathcal{Y}}}(\cdot \mid \boldsymbol{y})$ 下获得的条件概率为零。我们使用图 10 中的过程进行恢复，该过程接收当前目标字符串 $\boldsymbol{y}$、下一符号 y、当前剪枝阈值 τ以及超参数：最大重试次数 R、阈值下限 $\tau_{\min}$ 和最大驱逐深度 $D_{\max}$。每次重试时，τ 减半（下限为 $\tau_{\min}$，以防止 q 无限增长），并以指数递增的深度 $D = \min(2^{i-1}, D_{\max})$驱逐 $\boldsymbol{y}$ 前缀的缓存的分解，迫使 decompose 使用放宽后的阈值重新计算。随后恢复原始阈值。该过程是启发式的：无法保证经过固定次数的重试并将阈值减半后一定能够恢复。频繁回溯表明剪枝阈值过于激进；在这种情况下，反复驱逐和重新计算的开销可能会主导运行时间。在我们的实验中，我们使用$R = 20$、$\tau_{\min} = 10^{-10}$、$D_{\max} = 32$。实际中回溯并不频繁：当 $\tau \leq$ 1e-3时，所有实验中均未发生回退。在最粗的阈值（$\tau =$ 1e-1）下，每个段落的平均回退次数最高达到：$p_{\mathsf{gpt2}} \circ \mathrm{f}_{\alpha}$ 为 96 次（最大值 159），$p_{\mathsf{gpt2}} \circ \mathrm{f}_{\alpha} \circ \mathrm{f}_{\mathrm{ptb}}$ 为 5.1 次（最大值 18），$p_{\mathsf{dna}} \circ \mathrm{f}_{\mathrm{dna2aa}}$ 为 0.4 次（最大值 5）。

[25]确定性剪枝的一种替代方案是针对源字符串使用提议分布进行重要性采样；我们将这一方向留待未来工作。

120 def backtrack($\boldsymbol{y}, y, \tau, R, \tau_{\min}, D_{\max}$):
121 $\tau_0 \leftarrow \tau$122 for $i \in 1, \ldots, R$:
123 $\tau \leftarrow \max(\tau/2,\ \tau_{\min})$ 124 $D \leftarrow$
$\min(2^{i-1},\ D_{\max})$ 125 for $d \in 0, \ldots, D-1$:
126 if $d > |\boldsymbol{y}|$: break 127 驱逐缓存的
decompose($\boldsymbol{y}_{<|\boldsymbol{y}|-d}$) 128 $p \leftarrow$next_
dist($\boldsymbol{y}$) 129 if $p[y] > 0$: 130 $\tau \leftarrow \tau_0$
131
132 $\tau \leftarrow \tau_0$ return 133 引发
DeadEnd

图 10：回溯。每次重试时，将 τ 减半（下限为 $\tau_{\min}$），并逐步清除 $\boldsymbol{y}$ 的更长前缀（深度 $D = \min(2^{i-1}, D_{\max})$）对应的缓存分解，然后重新计算 next_dist。之后恢复 τ。

C.4 快速下一个符号分解 (分解_下一个)

由于 $\langle \mathcal{Q}(\boldsymbol{y}y) \rangle \subseteq \langle \mathcal{Q}(\boldsymbol{y}) \rangle$，我们有 $\mathcal{P}(\boldsymbol{y}y) = (\langle \mathcal{Q}(\boldsymbol{y}) \rangle \cap \langle \mathcal{Q}(\boldsymbol{y}y) \rangle) \sqcup \mathcal{R}(\boldsymbol{y}y)$。因此，可以通过从 $\mathcal{Q}(\boldsymbol{y}) \cup \mathcal{R}(\boldsymbol{y})$ 开始增量计算，得到 $\boldsymbol{y}y$ 的分解。其结构性原因是单调性：$\boldsymbol{y}$ 与 $\boldsymbol{y}y$ 的预覆盖具有相同的字母表和初始状态；二者仅接受状态不同，因为是否可接受取决于哪些缓冲区覆盖目标。这由前沿包含性质（命题 C.1）形式化说明。

该分解_下一个方法会同时计算 $\mathcal{Q}(\boldsymbol{y}y)$ 和 $\mathcal{R}(\boldsymbol{y}y)$ 对所有 $y \in \mathcal{Y}$ 的精确原像 $f^{-1}(\boldsymbol{y}y)$。与对每个分解($\boldsymbol{y}y$)分别调用相比，这是关键的效率提升 y。

广度优先搜索以 $\mathcal{Q}(\boldsymbol{y}) \cup \mathcal{R}(\boldsymbol{y})$ 为随机种子（第 152 行）。对于队列中的每个源字符串 $\boldsymbol{x}$，我们首先检查 is_exact_member($\boldsymbol{x}, \boldsymbol{y}$) 是否成立（第 156 行）；若成立，则将 $\boldsymbol{x}$ 记录到原像集合中。接下来，对于每个 $y \in$ reachable_outputs($\boldsymbol{x}, \boldsymbol{y}$)（定义见下文；第 159 行），我们将 $\boldsymbol{x}$ 分类为 $\boldsymbol{y}y$ 的柱集或成员；根据 f 的函数性，最多只有一个 y 可以是柱集（命题 C.4）。如果发现某个 y 对应柱集，则会吸收以 $\boldsymbol{x}$ 为根的整个子树——我们跳过所有符号的扩展（第 164 行）。否则，我们通过所有可达源符号进行扩展（第 165 行）。

这个分解_下一符号算法依赖接下来介绍的两种机制：用于快速计算前沿的目标递归，以及三个基于前沿的辅助函数。下面逐一介绍。

目标递归。调用is_柱集($\boldsymbol{x}$, $\boldsymbol{y}y$) 会在内部触发run$\boldsymbol{y}y$($\boldsymbol{x}$)。当父级前沿run$\boldsymbol{y}$($\boldsymbol{x}$) 已在缓存中时，会择机使用目标递归：扩展目标的前沿通过筛选得出，而无需重新计算整个源字符串链（命题C.1）。这是分解_下一符号中的典型情况，因为分解($\boldsymbol{y}$) 已经填充了缓存。此优化是透明的：is_柱集调用方式保持不变；逻辑完全封装在run中。

逐符号前沿辅助函数。BFS 循环使用三个基于前沿的辅助函数。

- is_精确_成员($\boldsymbol{x}$, $\boldsymbol{y}$) 用于检查 $\boldsymbol{x}$ 是否映射到 $\boldsymbol{y}$ 恰好，即is_精确_成员($\boldsymbol{x}$, $\boldsymbol{y}$) $\Longleftrightarrow (f(\boldsymbol{x}) = \boldsymbol{y})$。

- 可达_输入($\boldsymbol{x}$, $\boldsymbol{y}$) 保守地近似源符号集合 x'，使得将 $\boldsymbol{x}$ 扩展 x' 后仍与 $\boldsymbol{y}$兼容，即 $\{x' \in \mathcal{X} : \langle \boldsymbol{x}x' \rangle \cap \mathcal{P}(\boldsymbol{y}) \neq \emptyset\}$。伪代码会检查从前沿状态出发的转移，判断其扩展后的缓冲区是否与 $\boldsymbol{y}$前缀兼容；如果所得状态无法到达接受状态，结果可能会多报。
- 可达的_输出($\boldsymbol{x}, \boldsymbol{y}$) 保守地近似目标符号集合 y'，使得 $\boldsymbol{x}$ 的某个扩展可以生成 $\boldsymbol{y}y'$ 作为输出前缀，即 $\{y' \in \mathcal{Y} : \langle \boldsymbol{x} \rangle \cap \mathcal{P}(\boldsymbol{y}y') \neq \emptyset\}$。伪代码检查来自前沿状态的单步转移，这可能会过度报告（某个来自前沿状态、但无法到达接受状态的转移），因为前沿未经余修剪。需要多次转移才能扩展超过 $\boldsymbol{y}$ 的符号由扩展循环处理。

三者都直接根据前沿$\mathrm{run}_y(x)$计算：

134 def is_exact_member(x, y): 135 return $\exists (s, b) \in \mathrm{run}_y(x)$: 140 def reachable_outputs(x, y): 141 $\mathcal{F} \leftarrow \mathrm{run}_y(x)$136 $b = y \wedge s \in$ F 142 # y' 从 b （已提交）读取 137 def reachable_inputs($x\ y \in y\ x$ 143 ,): 138 return $\{x' \mid (s, b)$ run ( )139 $s \xrightarrow{x'\ y'} s' \in$ (:) T # 或 y 144 $by' \preceq y \vee y \preceq by'\}$ （边界） return $\{y' \mid (s, b) \in \mathcal{F}$,145 s $(s \xrightarrow{:y'} s') \in$T( ) $yy' \preceq by'\}$

扩展循环。 以下算法将上述随机种子、捷径和辅助函数整合为一个广度优先搜索，同时计算每个 $y \in \mathcal{Y}$ 的逐符号商和余项。

146 定义 decompose_next(y): 147 (Q, R) $\leftarrow$ decompose(y) 148 $\hat{y} \mapsto x$ 149 $\leftarrow \emptyset$ qs $\leftarrow$ {}; rs $\leftarrow$ {} # 映射 ps 集合 150 # 对于 y，余项不是柱集，因此对任何 yy 都不是（命题 C.3）151 non_cyl $\leftarrow$ R 152 q $\leftarrow$ Q $\cup$ R 153 while $|\mathrm{q}| > 0$: 154 q′ $\leftarrow \emptyset$155 for x in q: 156 if is_exact_member(x, y) : ps.add(x) # $f(x)$ 恰好 157 # 检查可达的 y；至多一个 y 能将 x 作为柱集（命题 C.4）。 158 $\hat{y} \leftarrow \bot$159 for y in reachable_outputs(x, y) : 160 if $\hat{y} = \bot$ and $x \notin$ non_cyl and is_cylinder(x, yy): 161 qs[y].add(x); $\hat{y} \leftarrow y$; continue 162 if is_member(x, yy): 163 rs[y].add(x) 164 if $\hat{y} \neq \bot$: continue # 已吸收 $\Rightarrow$，跳过对所有 y 的扩展 165 for $x' \in$ reachable_inputs(x, y) : 166 q′.add(xx') 167 q $\leftarrow$ prune(q′) 168 return (qs, rs, ps)

图 11：快速下一词元分解算法。

decompose_next(y) 会在单次广度优先搜索遍历中计算所有 $y \in \mathcal{Y}$ 对应的 $\mathcal{Q}(yy)$ 和 $\mathcal{R}(yy)$。is_cylinder(x, yy) 调用会触发 $\mathrm{run}_{yy}(x)$；由于父级前沿 $\mathrm{run}_y(x)$ 已缓存，因此可通过目标递归（命题 C.1）高效计算。对于 $\mathcal{R}(y)$ 中的种子，非圆柱捷径完全省去了全称性检查。广度优先搜索结束后，缓存已填充，后续的 decompose(yy) 调用无需耗费时间。精确原像也会作为副产品收集：如果 x 对某个 yy 是柱集，那么每个扩展产生的输出都严格长于 y，因此没有扩展能成为 y 的精确原像。

缓存填充。 广度优先搜索结束后，计算得到的 $\mathcal{Q}(yy)$ 和 $\mathcal{R}(yy)$ 应写入 decompose 的记忆化缓存中。后续的 decompose(yy) 调用（例如来自decompose_next(yy)）会命中缓存，而不会重新运行自身的广度优先搜索。

169**def** 分解_next(y): 170 …171 # 填充分解'的缓存，供所有子节点172 **for** $y \in$ qs.keys() $\cup$ rs.keys(): 173 缓存[分解 yy] $\leftarrow$ (qs[y], rs[y]) 174 **return** …

C.5 下一个符号分解捷径

该分解_next 算法（图11）通过一次广度优先搜索遍历，一次性为所有输出符号共享工作，而不是对每个符号调用分解，如分解 （图2）。从前一步的分解出发并使用目标递归，整体前沿计算得以复用。剩余瓶颈是逐符号的柱面检查is_柱集 （图9）。三项捷径可避免或降低该开销，每一项都由一条关于预覆盖结构的命题所支持。非柱面单调性与柱集唯一性使得在某些设定下可跳过广度优先搜索。组合普适性通过考察前沿如何划分为已提交元素与边界元素来确定额外符号。

(1) 非柱面捷径。 字符串 $\mathcal{R}(y)$ 不能是 y的商元素。而且，由于$\langle \mathcal{Q}(yy') \rangle \subseteq \langle \mathcal{Q}(y) \rangle$，它们也不是限制更严格的 yy 的商元素。因此，对这些种子跳过了耗时的柱面检查（第160行）（第151行）。

命题 C.3 (非柱面单调性)。 若 $x \in \mathcal{R}(y)$，则 $x \notin \mathcal{Q}(yy)$ 对所有 $y \in \mathcal{Y}$均成立。

证明。 我们有 $x \in \mathcal{R}(y)$ 蕴含 $\langle x \rangle \not\subseteq \mathcal{P}(y)$。由于 $\mathcal{P}(yy) \subseteq \mathcal{P}(y)$，因此 $\langle x \rangle \not\subseteq \mathcal{P}(yy)$，所以 $x \notin \mathcal{Q}(yy)$。 ■

(2) 柱集唯一性捷径。 由于我们考虑的变换都是函数，因此至多只有一个输出符号 y 可以具有 x 作为柱面元素。一旦找到一个柱面元素，就会对其余输出符号跳过柱面检查（第159行）。

命题 C.4 (柱集唯一性)。 对于函数 f 和目标前缀 y， $x \in \mathcal{Q}(yy) \cap \mathcal{Q}(yy') \implies y = y'$。

证明。 假设 $y \neq y'$ $x \in \mathcal{Q}\ yy\ \cap \mathcal{Q}\ yy'$ $\langle x \rangle \subseteq \mathcal{P}\ yy\ \cap \mathcal{P}\ yy' = f^{-1}\ yy\mathcal{Y}^* \cap$且 $f^{-1}\ yy'\mathcal{Y}^* = \emptyset$ $yy\mathcal{Y}^* \cap yy'\mathcal{Y}^* = \emptyset$ ()()。那么 ()()()()，其中最后一个等式成立，是因为是一个函数且。这与 $\langle x \rangle \neq \emptyset$矛盾。 ■

(3) 组合普适性捷径。给定 $x \in \mathcal{Q}(y)$ 和 $\widehat{y} \in \mathcal{Y}$，我们希望利用 y 处的覆盖前沿，判断$x \in \mathcal{Q}(y\widehat{y})$是否成立，而无需运行完整的幂集广度优先搜索。前沿运行(x, y)划分为已确定元素（s, b），其满足 $b \succeq y\widehat{y}$（累积输出已通过符号 $\widehat{y}$延伸至 y 之后）；以及边界元素（s, b），其满足 $b = y$ （下一个输出符号尚未确定）。仅靠已提交状态可能无法实现全称普遍性——某些输入符号可能没有从已提交状态出发的出边。边界状态可以填补这些缺口，前提是从每个边界状态出发的每条转移都以 $\widehat{y}$ 作为其下一个输出符号。

命题 C.5 (组合普适性)。 设 $x \in \mathcal{Q}(y)$， $\widehat{y} \in \mathcal{Y}$，并定义

$$S_c = \{s \mid (s, b) \in \mathsf{run}(x, y),\ b \succeq y\widehat{y}\},$$
$$S_b = \{s \mid (s, b) \in \mathsf{run}(x, y),\ b = y\}$$

若并集 $S_c \cup S_b$ 是全称的 （即 $[\![\mathsf{proj}_{\mathcal{X}}(\mathsf{f}[S_c \cup S_b])]\!] = \mathcal{X}^*$） ，则 $S_c \cap \mathsf{F} \neq \emptyset$，且从边界状态出发的每个转移都会产生 $\widehat{y}$，但 ε-输入转移可能产生 ε （即 $s \in S_b$和$s \xrightarrow{x:\,y} s' \in \mathsf{T}$ 蕴含 $y = \widehat{y}$或($x = \varepsilon \wedge y = \varepsilon$)） ，那么 $x \in \mathcal{Q}(y\widehat{y})$。

证明。设 $x' \in \mathcal{X}^*$ 任意。由 $S_c \cup S_b$的普遍性可知，存在 $(s, b) \in \mathsf{run}(x, y)$，满足$s \in S_c \cup S_b$，使得 s 在输入投影中对 x' 有一个接受运行。若 $s \in S_c$，则 $b \succeq y\widehat{y}$，因此相应路径在 f 中的输出已经以 $y\widehat{y}$开头，从而得到 $xx' \in \mathcal{P}(y\widehat{y})$。若 $s \in S_b$，则 $b = y$。根据互斥假设，从 s 出发的每个非-ε-输入转移都会产生输出 $\widehat{y}$，而 ε-输入转移则会产生 $\widehat{y}$或 ε。从 s 经由 ε:ε 弧可达的状态也属于 S_b（因为前沿在 ε下封闭，且缓冲区保持不变），因此它们满足相同的约束。因此，从 s 出发的任意路径上，第一个非-ε 输出都是 $\widehat{y}$，且输出以 $y\widehat{y}$开头，从而得到 $xx' \in \mathcal{P}(y\widehat{y})$。由于 x' 是任意的， $\langle x \rangle \subseteq \mathcal{P}(y\widehat{y})$，即 $x \in \mathcal{Q}(y\widehat{y})$。 ■

这条捷径与 IP 普遍性捷径（命题 C.6）互为补充：IP 普遍性是逐个状态的性质，而组合普适性是集合级性质，它利用了已提交状态和边界状态各自提供的部分覆盖。

该命题要求 $S_c \cup S_b$ （从这些状态开始的输入投影非确定性有限自动机接受 $\mathcal{X}^*$）具有集合层面的普遍性。实践中，我们采用更强但成本更低的充分条件：要求 $S_c \cup S_b$ 中的每个状态都属于预先计算好的 U。这需要 $O(|\mathcal{F}|)$ 时间，而集合层面的检查需要进行幂集广度优先搜索，其开销可能与完整的 is_柱集检查一样大；此捷径正是为了避免这种检查。

175 def combined_全称($\mathcal{F}, \boldsymbol{y}\widehat{y}$): 176 $S_c \leftarrow \{s \mid (s, \boldsymbol{b}) \in \mathcal{F}$, $\boldsymbol{b} \succeq \boldsymbol{y}\widehat{y}\}$ # 已覆盖 177 $S_b \leftarrow \{s \mid (s, \boldsymbol{b}) \in \mathcal{F}, \boldsymbol{b} = \boldsymbol{y}\}$ # 边界 178 if $\exists s \in S_c \cup S_b$: $s \notin$ U: return False 179 # 非-ε 输入必须产生 $\widehat{y}$; ε 输入可以产生 ε180 for $s \in S_b$: 181 for (_ $\xrightarrow{x}$: $\xrightarrow{y} s'$) ∈T(s): 182 if not ($y = \widehat{y}$ or ($x = \varepsilon \wedge y = \varepsilon$)): 183 return False 184 return $S_c \cap$F $\neq \emptyset$

C.6 输入投影全称性捷径(快速_普遍性_筛选)

回顾一下，柱面检查 会_柱集 验证源前缀的每个扩展是否都能映射到目标前缀之后；这需要使用可能代价高昂的幂集广度优先搜索。许多情况下，可以用一个廉价的充分条件来避免这种检查：如果从转换器的 $s \in$ S 启动后，其**输入投影是全覆盖的**，那么状态 s 就称为输入投影全称状态 $\mathcal{X}^*$——直观来说，无论后续输入是什么，转换器仍然能够生成输出。令 U $\subseteq$ S表示所有输入投影全称状态的集合。

命题 C.6 (IP 普遍性捷径)。 如果 $\boldsymbol{x}$ 相对于 $\boldsymbol{y}$ 的某个覆盖前沿状态是 *IP*-通用的，那么 $\boldsymbol{x}$ 是一个柱集： $\big(\exists (s, \boldsymbol{b}) \in \mathsf{run}(\boldsymbol{x}, \boldsymbol{y}) \colon \boldsymbol{b} \succeq \boldsymbol{y} \wedge s \in \mathsf{U}\big) \Longrightarrow \boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y})$。

证明。令 $(s, \boldsymbol{b}) \in \mathsf{run}(\boldsymbol{x}, \boldsymbol{y})$ 为前提中的见证，其中包含 $\boldsymbol{b} \succeq \boldsymbol{y}$ 和 $s \in$ U。则$[\![\mathrm{proj}_{\mathcal{X}}(\mathrm{fl}^{s})]\!] = \mathcal{X}^*$，因此对于任意 $\boldsymbol{x}' \in \mathcal{X}^*$，f 在 $\boldsymbol{x}\boldsymbol{x}'$ 上都存在一条接受运行，该运行经过 s，且输出前缀为 $\boldsymbol{y}$。因此 $\boldsymbol{x}\boldsymbol{x}' \in \mathcal{P}(\boldsymbol{y})$。由于 $\boldsymbol{x}'$ 是任意的，故 $\langle \boldsymbol{x} \rangle \subseteq \mathcal{P}(\boldsymbol{y})$，即 $\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y})$。 ■

图 12 展示了如何高效地预先计算输入投影全称状态集。我们首先对转换器进行输入投影，并移除 ε-转移（将每条路径 $s \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} s' \xrightarrow{x} s''$替换为直接转移 $s \xrightarrow{x} s''$；例如，参见 Hopcroft 等，2001，第 2 章），得到一个在 $\mathcal{X}$ 上的无 ε 非确定性有限自动机 A。这两种操作都保留状态集：输入投影会移除输出标签，而 ε-移除只会修改转移（通过 ε-闭包添加捷径），因此 $\mathsf{S}_{\mathsf{A}} = \mathsf{S}$ 和 U 中的状态都与前沿中的转导器状态直接对应。A 的状态 s 是输入投影全称状态，当且仅当它是接受状态，并且对于每个 $x \in \mathcal{X}$，至少有一个同样为输入投影全称状态的 x-后继——这是一个最大不动点条件。我们使用一种类似于 Kahn 拓扑排序算法（Kahn，1962）的算法在线性时间内计算该不动点：将候选集 U 初始化为 A 的接受状态，并维护每个状态针对每个符号的后继计数。当某个计数变为零时，就移除该状态，并通过反向转移索引将变化传播给其前驱。每条转移至多处理一次，因此时间复杂度为 $\mathcal{O}(|\mathsf{T}_{\mathsf{A}}|)$。

该捷径可以作为快速路径集成到中_柱集（图9）中中_柱集：

```
def is_柱集(x, y): # 记忆化 if fast_univ_filter(x, y):
return True # 快速路径   N ← |y|   ...   # 按照之前的
方式继续进行广度优先搜索
```

对于大多数接受状态都是输入投影全覆盖的转换器，这一捷径可避免在中_柱集中进行普遍性广度优先搜索，从而省去绝大多数柱检查。在 BPE 中，所有状态都是输入投影全覆盖，因此永远不需要进行普遍性广度优先搜索。

C.7 对 $p_{\mathcal{X}}$进行记忆化和批处理

源语言模型在两个地方会被查询：首先是在分解期间，概率质量剪枝会调用 $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})$ 对候选项进行排序；然后是在评分期间，prefix_prob 和 next_dist 会对商与余项上的 $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})$ 和 $p_{\mathcal{X}}(\boldsymbol{x})$ 求和。这些调用涉及许多相互重叠的源前缀。若采用朴素做法，每次调用都会从头重新计算整个链 $\overrightarrow{p_{\mathcal{X}}}(x_1) \cdot \overrightarrow{p_{\mathcal{X}}}(x_2 \mid x_1) \cdots$。对条件概率 $\overrightarrow{p_{\mathcal{X}}}(\cdot \mid \boldsymbol{x})$ 进行记忆化即可避免这一点：给定 $\boldsymbol{x}$ 的缓存状态后，每扩展一个符号 x，只需进行一次条件评估，无须重放整个历史。

对于 Transformer语言模型，缓存的状态就是键值（*KV*）缓存：自注意力层中累积的键张量和值张量。扩展一个词元时，会重用缓存的 KV 条目，并针对新位置执行一次前向传播。此外，大多数语言模型在一次调用中会返回完整的下一符号分布 $\overrightarrow{p_{\mathcal{X}}}(\cdot \mid \boldsymbol{x})$，因此我们缓存完整分布，而不是

185 # 预处理：输入投影并移除 ε 186 A $\leftarrow$ 移除eps（proj$_{\mathcal{X}}$(f)） 187 # 基于队列的最大不动点 188 $\leftarrow$ $_{A}$ $\leftarrow \emptyset$ U F； q 计数 190 for ($s \xrightarrow{x} s'$) $\in$T$_A$ with $s' \in$ 189 $[\cdot,\cdot] \leftarrow 0$
U: 191 计数$[s, x]$ += 1192 for $s \in$ U: 193 if $\exists x \in \mathcal{X}$: 计数$[s, x] = 0$: 194 q.add($s$) 195 while q $\neq \emptyset$: 196 $s' \leftarrow$ q.pop() 197 U $\leftarrow$ U $\setminus \{s'\}$198 for (s, x) with ($s \xrightarrow{x} s'$) $\in$ T$_A$: 199 计数$[s, x]$ -= 1200 if 计数$[s, x] = 0$ and $s \in$ U: 201 q.add(s) 202 def fast_全称_过滤器($\boldsymbol{x}, \boldsymbol{y}$): 203 # 是否存在具有全称输入投影的覆盖前沿状态？ 204 return $\exists(s, \boldsymbol{b}) \in$ run($\boldsymbol{x}, \boldsymbol{y}$): $\boldsymbol{b} \succeq \boldsymbol{y} \wedge s \in$ U

图 12：高效计算输入投影全称状态以及快速_univ_过滤方法。

各个值。当扩展循环生成子节点 $\boldsymbol{x}x_1, \ldots, \boldsymbol{x}x_k$时，它们的前缀概率会直接从缓存分布中读取，无需额外调用语言模型。[26]

剪枝时进行批处理。剪枝步骤(§C.3) 对队列中的每个字符串都需要 $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})$。队列中的许多字符串共享一个公共前缀，但以不同符号扩展。我们按父节点对队列分组：对于每个其下一符号分布 $\boldsymbol{x}$ $\boldsymbol{x}_{<N}$ 尚未缓存的父节点 $\overrightarrow{p_{\mathcal{X}}}(\cdot \mid \boldsymbol{x}_{<N})$，我们发起一次批量语言模型调用。每次调用都会返回完整分布，从中可以同时读取所有子节点的$\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}_{<N}x) = \overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}_{<N}) \cdot \overrightarrow{p_{\mathcal{X}}}(x \mid \boldsymbol{x}_{<N})$。对于基于GPU的语言模型，将这些父节点上下文批处理到一次前向传播中，可以摊销单次调用开销。

C.8 实现自回归接口

给定该分解，第 §C.1 节中的转导语言模型接口实现如下。前缀概率 $\overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y})$ 对 Q 上的语言模型前缀概率以及 R 上的语言模型字符串概率求和。字符串概率 $p_{\mathcal{Y}}(\boldsymbol{y})$ 在精确原像上求和。下一符号分布 $\overrightarrow{p_{\mathcal{Y}}}(\cdot \mid \boldsymbol{y})$ 通过decompose_next计算，该过程在一次广度优先搜索遍历中提供每个符号的分解和精确原像；计算成本主要由 decompose_next决定，而一旦获得每个符号的商、余项和原像，评分就可简化为查询记忆化源语言模型（§C.7）。

209 def 前缀_prob($\boldsymbol{y}$): 210 (Q, R) $\leftarrow$ 分解($\boldsymbol{y}$) 211 return $\sum_{\boldsymbol{x} \in}$Q $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}) + \sum_{\boldsymbol{x} \in}$ R $p_{\mathcal{X}}(\boldsymbol{x})$ 212 def prob($\boldsymbol{y}$): 213 (_,_, ps) $\leftarrow$ 分解_next($\boldsymbol{y}$) 214 return $\sum_{\boldsymbol{x} \in}$ ps $p_{\mathcal{X}}(\boldsymbol{x})$

215 def next_dist($\boldsymbol{y}$): 216 (qs, rs, ps) $\leftarrow$ 分解_next($\boldsymbol{y}$) 217 $p \leftarrow \{\}$; $Z \leftarrow$ **前缀_prob($\boldsymbol{y}$) 218 for** $\boldsymbol{x} \in$ **ps:** p[**EOS**] += $p_{\mathcal{X}}(\boldsymbol{x})/Z$**219 for** $y \in$ **qs.keys()** $\cup$ **rs.keys(): 220 for** $\boldsymbol{x} \in$ **qs**[y]: $p[y]$ += $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})/Z$**221 for** $\boldsymbol{x} \in$ **rs**[y]: $p[y]$ += $p_{\mathcal{X}}(\boldsymbol{x})/Z$**222 return** p

图 13：使用分解_next快速实现

[26]我们的实现使用 genlm (https://github.com/genlm/genlm-backend)，以提供无缝的缓存和自动批处理接口。

作为参考，朴素的下一符号分布会针对每个 前缀_prob($\boldsymbol{y}y$)分别独立计算 y (见 §C.1）。参考实现是正确的，但效率低下，因为它没有共享计算：每个 前缀_prob($\boldsymbol{y}y'$) 调用都会触发各自的 分解 广度优先搜索，而且无法利用 §C.4 中所述的非圆柱捷径、目标递归或缓存填充。

C.9 全称快速路径

当每个转导器状态都是 IP-通用时，余项始终为空，且每个新发现的元素都会立即成为商元素，因此分解会在单个广度优先搜索轮次内终止。此外，如果每条非ε输入弧至少产生一个输出符号，那么就可以通过预先计算的首输出表，以闭式形式解析边界前沿元素。这两个条件结合起来，使 next_dist(图 13) 得以简化为 全称_通用_next_dist (图 14)。下面介绍这些简化（归一化保持不变）。

分解（简化版）。由于每个状态都是 IP-通用，$\mathcal{R}(\boldsymbol{y}) = \emptyset$ 对所有 $\boldsymbol{y}$都成立：预覆盖中的每个源前缀都属于商。对 decompose($\boldsymbol{y}$) (图 2) 的调用会在单个广度优先搜索轮次内终止——每个新发现的元素都会立即被分类为商元素。因此，decompose_next 不再需要：逐符号分类已并入下文的评分过程。

分数（简化版）。对于每个$(\mathcal{F}, \boldsymbol{x}) \in \mathtt{Q}$，前沿划分为已确定元素$(s, \boldsymbol{b})$，其中 $\boldsymbol{b} \succ \boldsymbol{y}$，以及边界元素，其中 $\boldsymbol{b} = \boldsymbol{y}$，如组合普适性捷径（§C.5）所示。已确定元素已经确定了它们的下一个输出符号 $\widehat{y} = \boldsymbol{b}_{N+1}$；由于所有状态都是 IP-通用的，每个元素都会直接为 $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})$ 贡献 $p[\widehat{y}]$；因此无需展开。

边界元素（$\boldsymbol{b} = \boldsymbol{y}$）还需要一个源符号才能确定输出。无$\varepsilon$输出前提确保每条非$\varepsilon$输入弧至少产生一个输出符号，因此预先计算的首输出表$\mathtt{fo}(S, x)$（在伪代码中定义）将每个输入符号映射到给定幂状态下由其产生的唯一输出。该符号是唯一的：f 的函数性要求来自状态 S 的所有转移在 x 上产生相同的输出，因为 IP-通用性保证每个后继都有一个接受延续（因此两个输出都会被实现）。这样，每个边界元素只需一次批量语言模型调用即可处理：每个商元素查询一次 $\ell = \overrightarrow{p_{\mathcal{X}}}(\cdot \mid \boldsymbol{x})$，并且对于每个 $x \in \mathcal{X}$，质量 $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}) \cdot \ell(x)$ 都归于 $\mathtt{fo}(S, x)$。如果任何边界状态是接受状态，语言模型的 EOS 概率 $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}) \cdot \ell(\text{EOS})$ 将计入 $p[\text{EOS}]$。

该$\mathtt{f}_\alpha$ 转换器满足两个前置条件（表 4：所有状态均为 IP-通用，且非 ε- 输出弧上没有ε- 输入弧），这解释了它们显著更高的吞吐量（§7）。DNA 转导器 $\mathtt{f}_{\text{dna2aa}}$ 的所有状态均为 IP-通用，但它使用 ε-输出弧（每个密码子的前两个碱基不输出氨基酸），因此快速路径不适用。

```
223 def all_全称_next_dist(y): 224 (Q_)
 ← 分解(y) # R = ∅ 总是 225    p ← {};  Z ←
前缀_prob(y) 226 for (F, x) in Q: 227 #
已提交：输出目标之外的内容 228 for
 ŷ ∈ {b_{N+1} | (s, b) ∈ F, b ≻ y}:
229          p[ŷ] += p_X→(x) 230 # 边界：输出 =
目标 231       S_b ← {s | (s, b) ∈ F, b = y}
232 if  S_b ≠ ∅: 233          ℓ ← p_X→(· | x) # 一
次语言模型调用 234 for  x ∈ X:
235          ŷ ← fo(S_b, x) 236 if  ŷ ≠ ⊥:
237            p[ŷ] += p_X→(x) · ℓ(x) 238 if
 ∃ s ∈ S_b :  s ∈ F: 239          p[EOS] += p_X→
(x) · ℓ(EOS) 240 return  p/Z

241 def fo(S, x): 242 for (_ --x:y--> s')
 ∈ T(S, x): 243       # 若为函数式则
唯一 &244       # 所有状态均为 IP-通
用245       return y246    return ⊥
```

图 14：全称_全称_后继_分布：当所有状态都是 IP- 全称且每条非ε输入弧都会产生输出时，后继_分布（图 13）的特例。$\mathrm{R} = \emptyset$ 始终如此，因此只有 Q 会产生贡献。边界元素通过 fo 解析，每个商元素都需要调用一次语言模型。

D 有限性条件的证明

D.1 前缀单调映射的性质

命题 6.1。 以下各项等价：*(i)* f 是前缀单调的 *(ii)* $f(\langle \boldsymbol{x} \rangle) \subseteq \langle f(\boldsymbol{x}) \rangle$ 对所有 $\boldsymbol{x} \in \mathcal{X}^*$ *(iii)* $\mathcal{P}(f(\boldsymbol{x})) = \mathcal{C}(f(\boldsymbol{x}))$ 对所有 $\boldsymbol{x} \in \mathcal{X}^*$ *(iv)* f 是前缀连续的。

证明。 $(1) \Rightarrow (2)$：前缀单调性意味着，对于任意 $\boldsymbol{x}$， $\boldsymbol{x}' \in \mathcal{X}^*$，若有 $\boldsymbol{x} \preceq \boldsymbol{x}\boldsymbol{x}'$ 则 $f(\boldsymbol{x}) \preceq f(\boldsymbol{x}\boldsymbol{x}')$，因此存在一个 $\boldsymbol{y} \in \mathcal{Y}^*$ 使得 $f(\boldsymbol{x}\boldsymbol{x}') = f(\boldsymbol{x})\boldsymbol{y}$。又由于 $\boldsymbol{x}'$ 是任意选取的，(2) 成立。

(2) $\Rightarrow$ (3)：令 $\boldsymbol{x} \in \mathcal{X}^*$。假设 $f(\langle \boldsymbol{x} \rangle) \subseteq \langle f(\boldsymbol{x}) \rangle$。那么， $\langle \boldsymbol{x} \rangle \subseteq f^{-1}(f(\langle \boldsymbol{x} \rangle)) \subseteq \mathcal{P}(f(\boldsymbol{x}))$。注意，对于任意 $\boldsymbol{x}' \in \mathcal{P}(f(\boldsymbol{x}))$， $\mathcal{P}(f(\boldsymbol{x}')) \subseteq \mathcal{P}(f(\boldsymbol{x}))$。因此， $\langle \boldsymbol{x}' \rangle \subseteq \mathcal{P}(f(\boldsymbol{x}')) \subseteq \mathcal{P}(f(\boldsymbol{x}))$，于是 $\boldsymbol{x}' \in \mathcal{C}(f(\boldsymbol{x}))$。这表明 $\mathcal{P}(f(\boldsymbol{x})) = \mathcal{C}(f(\boldsymbol{x}))$。

(3) $\Rightarrow$ (4)：根据假设，预覆盖中的任意元素都是商元素的扩展，因此也是商元素柱集中的成员。

(4) $\Rightarrow$ (1)：假设 f 是前缀连续的，即对所有 $\boldsymbol{y}$， $\mathcal{P}(\boldsymbol{y})$ 都是柱状的。令 $\boldsymbol{x} \preceq \boldsymbol{x}'$。由于 $f(\boldsymbol{x}) \preceq f(\boldsymbol{x})$，因此有 $\boldsymbol{x} \in \mathcal{P}(f(\boldsymbol{x}))$。根据前缀连续性， $\mathcal{P}(f(\boldsymbol{x}))$ 是柱状的，因此 $\langle \boldsymbol{x} \rangle \subseteq \mathcal{P}(f(\boldsymbol{x}))$。特别地， $\boldsymbol{x}' \in \mathcal{P}(f(\boldsymbol{x}))$，由此得到 $f(\boldsymbol{x}) \preceq f(\boldsymbol{x}')$。■

D.2 严格前缀单调映射的商集界

下面的命题给出了映射为严格前缀单调时商的大小上界。**命题 D.1**。 设 $f:\ \mathcal{X}^* \to \mathcal{Y}^*$ 是一个严格前缀单调映射。那么，对每个 $\boldsymbol{y} \in \mathcal{Y}^*$:

1. $f^{-1}(\boldsymbol{y}) = \mathcal{Q}(\boldsymbol{y}) \setminus \bigsqcup_{y\in\mathcal{Y}} \mathcal{Q}(\boldsymbol{y}y)$
2. $\bigsqcup_{y\in\mathcal{Y}} \mathcal{Q}(\boldsymbol{y}y) \subseteq \mathcal{Q}(\boldsymbol{y})\,(\mathcal{X} \sqcup \{\varepsilon\})$
3. $|\mathcal{Q}(\boldsymbol{y})| \le (|\mathcal{X}|+1)^{|\boldsymbol{y}|}$

证明。 (1) 注意到（利用命题 6.1），对于每个 $\boldsymbol{y} \in \mathcal{Y}^*$

$$\mathcal{C}(\boldsymbol{y}) = \mathcal{P}(\boldsymbol{y}) = f^{-1}(\boldsymbol{y}) \sqcup \bigsqcup_{y\in\mathcal{Y}} \mathcal{P}(\boldsymbol{y}y) = f^{-1}(\boldsymbol{y}) \sqcup \bigsqcup_{y\in\mathcal{Y}} \mathcal{C}(\boldsymbol{y}y) \tag{11}$$

特别地， $\mathcal{Q}(\boldsymbol{y})$ 在 $\mathcal{P}(\boldsymbol{y})$ 中的极小性意味着

$$\mathcal{Q}(\boldsymbol{y}) \cap \bigsqcup_{y\in\mathcal{Y}} \mathcal{Q}(\boldsymbol{y}y) = \mathcal{Q}(\boldsymbol{y}) \cap \bigsqcup_{y\in\mathcal{Y}} \langle \mathcal{Q}(\boldsymbol{y}y)\rangle \tag{12}$$

因此，我们得到恒等式 $f^{-1}(\boldsymbol{y}) = \mathcal{C}(\boldsymbol{y}) \setminus \bigsqcup_{y\in\mathcal{Y}} \mathcal{C}(\boldsymbol{y}y) = \langle \mathcal{Q}(\boldsymbol{y})\rangle \setminus \bigsqcup_{y\in\mathcal{Y}} \langle \mathcal{Q}(\boldsymbol{y}y)\rangle$，进而得到包含关系

$$\mathcal{Q}(\boldsymbol{y}) \setminus \bigsqcup_{y\in\mathcal{Y}} \mathcal{Q}(\boldsymbol{y}y) = \mathcal{Q}(\boldsymbol{y}) \setminus \bigsqcup_{y\in\mathcal{Y}} \langle \mathcal{Q}(\boldsymbol{y}y)\rangle \subseteq f^{-1}(\boldsymbol{y}) \tag{13}$$

对于反向包含，设 $\boldsymbol{x}' \in f^{-1}(\boldsymbol{y})$。那么存在唯一的 $\boldsymbol{x}_q \in \mathcal{Q}(\boldsymbol{y})$，使得 $\boldsymbol{x}_q \preceq \boldsymbol{x}'$，这由商集的定义可知。由于 f 是严格前缀单调的，且 $f(\boldsymbol{x}_q) \preceq f(\boldsymbol{x}') = \boldsymbol{y}$，我们必有 $\boldsymbol{x}_q = \boldsymbol{x}'$。这表明 $f^{-1}(\boldsymbol{y}) \subseteq \mathcal{Q}(\boldsymbol{y})$，从而完成了 (1) 的证明。

(2) 固定 $y \in \mathcal{Y}$， $\boldsymbol{y} \in \mathcal{Y}^*$ 并令 $\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y}y)$。存在唯一的 $\boldsymbol{x}_y \in \mathcal{Q}(\boldsymbol{y})$ 使得 $\boldsymbol{x}_y \preceq \boldsymbol{x}$。将 $\langle \boldsymbol{x}_y\rangle$与（11）两边相交，可得

$$\langle \boldsymbol{x}_y\rangle = (\langle \boldsymbol{x}_y\rangle \cap f^{-1}(\boldsymbol{y})) \sqcup \bigsqcup_{y'\in\mathcal{Y}} (\langle \boldsymbol{x}_y\rangle \cap \mathcal{C}(\boldsymbol{y}y')) \tag{14}$$

$$= (\langle \boldsymbol{x}_y\rangle \cap f^{-1}(\boldsymbol{y})) \sqcup \langle \boldsymbol{x}\rangle \sqcup C_{\boldsymbol{x}} \tag{15}$$

其中 $C_{\boldsymbol{x}} \text{ 定义为}= \bigsqcup_{y'\in\mathcal{Y}} (\langle \boldsymbol{x}_y\rangle \cap \mathcal{C}(\boldsymbol{y}y')) \setminus \langle \boldsymbol{x}\rangle$。注意，该集合是柱状的。由严格单调性可得 $|\langle \boldsymbol{x}_y\rangle \cap f^{-1}(\boldsymbol{y})| \le 1$。因此有以下两种情况：(i) $\langle \boldsymbol{x}_y\rangle \cap f^{-1}(\boldsymbol{y}) = \emptyset$，此时

$$\boldsymbol{x} = \boldsymbol{x}_y \in \mathcal{Q}(\boldsymbol{y}) \cap \mathcal{Q}(\boldsymbol{y}y),$$

或 (ii) $\langle \boldsymbol{x}_y\rangle \cap f^{-1}(\boldsymbol{y}) = \{\boldsymbol{x}_y\}$。那么存在一个单词 $z \in \mathcal{X}^*$ 使得 $\boldsymbol{x} = \boldsymbol{x}_y z$。令 $z' \prec z$。若 $z' \ne \varepsilon$，则 $\boldsymbol{x}_y z' \in C_{\boldsymbol{x}}$。由于 $C_{\boldsymbol{x}}$ 是柱状的，它包含 $\langle \boldsymbol{x}_y z'\rangle$，特别是必定包含$\langle \boldsymbol{x}\rangle$，这与 $C_{\boldsymbol{x}}$的定义矛盾。因此不存在非空真前缀 $z' \prec z$，所以 z 是一个单独符号 $x \in \mathcal{X}$ 且

$$\boldsymbol{x} = \boldsymbol{x}_y x \in \mathcal{Q}(\boldsymbol{y})\mathcal{X}.$$

（3）对于任意 $\boldsymbol{y} \in \mathcal{Y}^*$ 以及任意 $y \in \mathcal{Y}$，由（2）可得，

$$|\mathcal{Q}(\boldsymbol{y}y)| \le (|\mathcal{X}|+1)|\mathcal{Q}(\boldsymbol{y})| \tag{16}$$

沿任意字符串 $\boldsymbol{y} \in \mathcal{Y}^*$迭代此界，我们得到:

$$|\mathcal{Q}(\boldsymbol{y})| \le (|\mathcal{X}|+1)^{|\boldsymbol{y}|}|\mathcal{Q}(\varepsilon)| = (|\mathcal{X}|+1)^{|\boldsymbol{y}|} \qquad \blacksquare$$

引理 6.1 给出了推导有限预覆盖分解的充分条件。此类转换器的一个示例见例3。

引理 6.1. 设 $f\colon\ \mathcal{X}^* \to \mathcal{Y}^*$ 是由转换器 f 实现的函数。分解$(\mathcal{Q}(y)\ \mathcal{R}(y))$ 对每个 $y \in \mathcal{Y}^*$ 都是有限的，如果：

(i) 无 ε输出环：f 不包含每条弧都输出 ε的环。*(ii)* 安全性：f 的每个状态都是安全的，归纳定义为满足下述条件的最小集合：若 s 满足以下条件则为安全：*(a)* s是 *IP*-通用的；*(b)* $|[\![\mathsf{f}_{[s]}]\!]| < \infty$ （有限闭包）；或 *(c)* 对所有转移 $s \xrightarrow{x:y} s'$， s' 是安全的。

证明。固定 $y \in \mathcal{Y}^*$；我们证明 $\mathcal{Q}(y)$与 $\mathcal{R}(y)$是有限的。设 Π_y 为 f 中恰好输出 y的所有路径的集合，即 $\Pi_y \overset{\text{def}}{=} \{s_0 \xrightarrow{x_1:y_1} s_1 \cdots s_{N-1} \xrightarrow{x_N:y_N} s_N \mid s_0 \in \mathsf{I}\ y_1 \cdots y_N = y\}$。由条件 (i)，$\mathsf{f}$ 的 ε输出子图是无环的，因此任何不产生输出的子路径长度至多为 $|\mathsf{S}|$。由于总输出为 y，每条路径的长度有界，且 Π_y 是有限的。设 $\Pi_{\succeq y}$ 为通过扩展 Π_y 中的根、直至到达满足安全性基础情形的状态而形成的所有有效路径的集合。由条件 (ii)，f 的每个状态都是安全的——特别是 Π_y中路径的终止状态。这意味着任何扩展路径都不能在不满足基础情形（IP-普遍性或有限闭包）的情况下无限延续。由于有限状态转换器具有有限分支，且不存在无限的有效扩展路径，K˝onig 引理蕴含扩展树是有限的。[27] 因此，$\Pi_{\succeq y}$ 是一个有限集。

令 $\mathsf{S}_U \overset{\text{定义}}{=} \{s \in \mathsf{S}\colon\ s \text{是 IP-通用}\}$，并令 $\mathsf{S}_C \overset{\text{定义}}{=} \{s \in \mathsf{S}\colon\ |[\![\mathsf{f}_{[s]}]\!]| < \infty\}$ 分别表示满足条件 (a) 和 (b) 的状态集合，并令 $\mathsf{S}_S \overset{\text{定义}}{=} \mathsf{S}_U \cup \mathsf{S}_C$。我们可以根据终止状态将扩展路径集合不相交地分解为：

$$\Pi_{\succeq y} = \left\{\rho \in \Pi_{\succeq y}\colon \rho_{|\rho|} \in \mathsf{S}_U\right\} \sqcup \left\{\rho \in \Pi_{\succeq y}\colon \rho_{|\rho|} \in \mathsf{S}_C\right\} \tag{17}$$

$$= \left\{\boldsymbol{\pi} \cdot s_{|\boldsymbol{\pi}|} \xrightarrow{\cdots} s_N \mid \boldsymbol{\pi} \in \Pi_y, s_N \in \mathsf{S}_U, \{s_{|\boldsymbol{\pi}|}, \ldots, s_{N-1}\} \cap \mathsf{S}_S = \emptyset\right\}$$

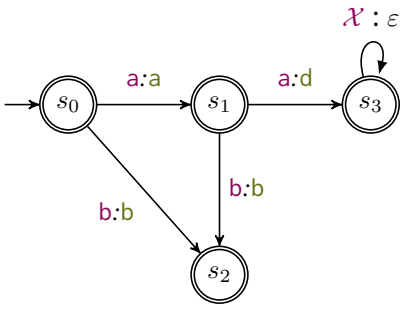
$$\sqcup \left\{\boldsymbol{\pi} \cdot s_{|\boldsymbol{\pi}|} \xrightarrow{\cdots} s_N \mid \boldsymbol{\pi} \in \Pi_y, s_N \in \mathsf{S}_C, \{s_{|\boldsymbol{\pi}|}, \ldots, s_{N-1}\} \cap \mathsf{S}_S = \emptyset\right\} \tag{18}$$

第一项根据定义对应于 $\mathcal{Q}(y)$。由于 $\Pi_{\succeq y}$ 是有限的，因此该集合也是有限的。第二项收集了通向状态 $s \in \mathsf{S}_C$的路径。余项 $\mathcal{R}(y)$ 由这些路径对应的输入字符串与 s所接受的有限语言拼接而成。由于路径数量有限，且每个停止状态的闭包都是有限的，$\mathcal{R}(y)$ 是有限的。■

示例 *3*。考虑下面这个在 $\mathcal{X} = \{a, b\}$, $\mathcal{Y} = \{a,b,d\}$ 上的转换器，它编码了一个（部分）函数 f。这四个状态展示了安全性的所有情形。状态 s_3 是全称的：它是接受状态，并且对每个输入符号都有一个以 ε 为输出的自环，因此 $L(s_3) = \mathcal{X}^*$ （基础情形 *a*）。状态 s_2 具有有限闭包：它是接受状态且没有出边，因此 $L(s_2) = \{\varepsilon\}$ （基础情形 *b*）。状态 s_1 既不是全称的，也不具有有限闭包，但根据递归步骤 *(c)* 是安全的：它的后继状态 s_3 （经由 *a*）和 s_2 （经由 *b*）都安全。类似地， s_0 是安全的，因为它的后继状态 s_1 （经由 *a*）和 s_2 （经由 *b*）都安全。

a:a 对于目标 $y = a$，路径集 $\Pi_a = \{s_0 \longrightarrow s_1\}$ 是有限的（条件 *i*），且终止状态 s_1 是安全的（条件 *ii*）。从 s_1扩展：输入 *a* 到达全称状态 s_3，为 $\mathcal{Q}(a)$贡献 *aa*；输入 *b* 到达有限闭包状态 s_2，为 $\mathcal{R}(a)$贡献 *ab*。根字符串 *a* 也属于 $\mathcal{R}(a)$：s_1 是接受状态，且 $f(a) = a \succeq a$，但 *a* 不是柱状的，因为 *ab* 无法进一步扩展（卡在 s_2）。因此， $\mathcal{Q}(a) = \{aa\}$，而 $\mathcal{R}(a) = \{a,\ ab\}$。

[27] K˝onig 引理指出，每棵有限分支的无限树都必定有一条无限路径。由于我们的字母表是有限的（有限分支），且安全性条件禁止非基例状态之间出现循环（不存在无限路径），因此得到的扩展树必定是有限的。

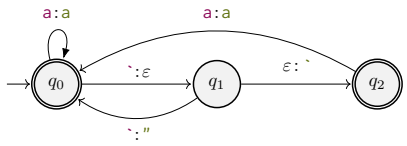

$\mathcal{X} : \varepsilon$
a:a
a:d
s_0
s_1
s_3
b:b
b:b
s_2

E 分解示例

我们给出三个分解和前缀概率的具体示例。

示例 4。 下面的转换器将连续的反引号（``）合并为引号（″），但其他符号 a 和 ` 保持不变。它可以被视为文本规范化的一个最小示例。

$$\mathsf{f} = \left[\quad\right] \tag{19a}$$



在计算前缀概率 $\overrightarrow{p_{\mathcal{Y}}}(\texttt{`})$ 时，以两个反引号开头的序列会被映射为引号，因此不会出现在预覆盖中。相反，单个反引号会保留在余项中，而反引号后跟 a 的序列会进入商中，因为任何进一步的扩展都会保留初始反引号。因此，我们有（通过直观地消去被映射为引号的后续部分），

$$\mathcal{P}(\texttt{`}) = \{\texttt{`}, \cancel{\texttt{``}}, \cancel{\texttt{```}}, \cancel{\texttt{````}}, \ldots\} \cup \{\texttt{`a}, \texttt{`a`}, \texttt{`aa}, \ldots\} \tag{19b}$$

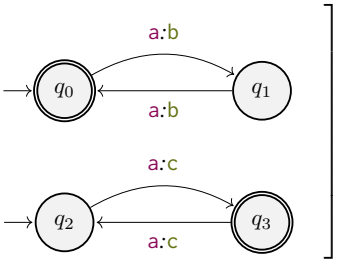
$$= \{\texttt{`}\} \sqcup \{\texttt{`a}\}\,\mathcal{X}^* \tag{19c}$$

因此，$\overrightarrow{p_{\mathcal{Y}}}(\texttt{`}) = p_{\mathcal{X}}(\texttt{`}) + \overrightarrow{p_{\mathcal{X}}}(\texttt{`a})$。类似地，我们有 $\mathcal{P}(\texttt{″}) = \{\texttt{``}\}\mathcal{X}^*$，因此 $\overrightarrow{p_{\mathcal{Y}}}(\texttt{″}) = \overrightarrow{p_{\mathcal{X}}}(\texttt{``})$。

示例 4 展示了使用余项和商的高效性；这两个集合都只有一个元素，却完整地规定了哪些概率会对 $\mathcal{P}(\texttt{`})$ 产生贡献。然而，也存在一些不适合进行如此高效分解的边界情况。

例5。 考虑以下映射 f 及其作为转换器的表示f：

$$f(\mathsf{a}^n) = \mathsf{b}^n \text{ 如果 } n \text{ 是偶数 否则 } \mathsf{c}^n \qquad \mathsf{f} = \left[\quad\right] \tag{20a}$$



其中 $\mathcal{X} = \{\mathsf{a}\}$ 和 $\mathcal{Y} = \{\mathsf{b}, \mathsf{c}\}$。b 的预覆盖如下

$$\mathcal{P}(\mathsf{b}) = \{\cancel{\varepsilon}, \cancel{\mathsf{a}}, \mathsf{aa}, \cancel{\mathsf{aaa}}, \mathsf{aaaa}, \cancel{\mathsf{aaaaa}}, \mathsf{aaaaaa}, \ldots\} \tag{20b}$$

遗憾的是，这个例子对我们的方法而言颇具挑战性，因为余项集不是有限集：

$$\mathcal{R}(\mathsf{b}) = \left\{\mathsf{a}^{2^n} \mid n > 0\right\}, \mathcal{Q}(\mathsf{b}) = \emptyset \tag{20c}$$

换句话说，这个映射从不解除偶数约束，因此我们永远在求和。

最后，我们在例 6 中给出一个只有一个余项元素的简单转换器示例。**例6**。 考虑以下映射 f，如图 *15* 所示。

$$f(\mathsf{a}^n) = \mathsf{b}^n \textit{ if } n \neq 2 \textit{ else } \mathsf{c} \tag{21a}$$

$\mathcal{X} = \{\mathsf{a}\}$ 和 $\mathcal{Y} = \{\mathsf{b}, \mathsf{c}\}$。假设我们要计算前缀概率 $\overrightarrow{p_{\mathcal{Y}}}(\mathsf{b})$。

$$\mathcal{P}(\mathsf{b}) = \{\mathsf{a}, \cancel{\mathsf{aa}}, \mathsf{aaa}, \mathsf{aaaa}, \mathsf{aaaaa}, \ldots\} \tag{21b}$$

$$\mathcal{R}(\mathsf{b}) = \{\mathsf{a}\}, \mathcal{Q}(\mathsf{b}) = \{\mathsf{aaa}\} \tag{21c}$$

因此，

$$\overrightarrow{p_{\mathcal{Y}}}(\mathsf{b}) = \sum_{\boldsymbol{x} \in \mathcal{R}(\mathsf{b})} p_{\mathcal{X}}(\boldsymbol{x}) + \sum_{\boldsymbol{x} \in \mathcal{Q}(\mathsf{b})} \overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}) \tag{21d}$$

$$= p_{\mathcal{X}}(\mathsf{a}) + \overrightarrow{p_{\mathcal{X}}}(\mathsf{aaa}) \tag{21e}$$

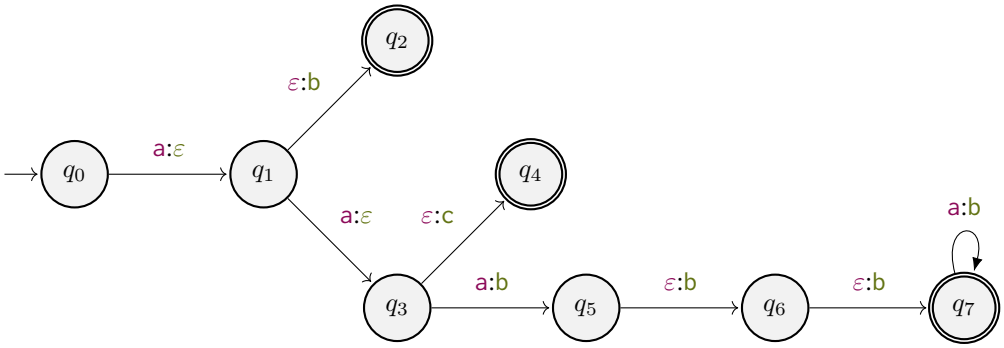


图15：一个有限状态转换器，将 n 个“a”的出现映射为相同数量的“b”，但当输入恰好是两个“a”时，则将其映射为“c”。

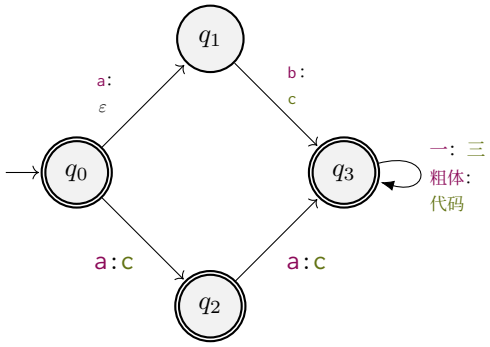


图 16：一个有限状态转换器，其中 ε-输出弧 q_0 a:$\xrightarrow{\varepsilon}$ q_1 会生成一个前沿元素，其输出尚未到达目标。对于目标 $\boldsymbol{y}$ = c 和源前缀 $\boldsymbol{x}$ = a，前沿为 $\mathcal{F}$ ={(q_1, ε), (q_2, c)}。覆盖状态 $\{q_2\}$ 并非输入投影全覆盖（q_2 在 b 上没有弧），因此快速 _univ_filter(a, c) 会失败，并将 a 错误地分类为非商。然而， q_1 会处理输入 b 并产生 c，与目标相匹配。_cylinder 中的广度优先搜索确认了整个前沿上的普遍性，正确地将 a 归入 $\mathcal{Q}$(c)。

F 实验设置

此处详细介绍了复现 §7 和 §G.5 中实验的实验设置。

F.1 数据集

对于 §7 中的令牌到字节和 PTB 实验，我们从 Hugging Face datasets 库中选取 `wikitext-2-raw-v1` 数据集测试集的前 10 个段落（不包括标题）（对应前 7684 字节）（Merity 等人，2017）。我们使用同一数据集和划分中的前 256 字节来运行 §G.5 中的实验。对于 §7 中的 DNA 实验，我们采样了 65 种人类蛋白质[28]，，每种由 4 至 12 个氨基酸组成，其登录号见表 1。

表 1：本研究使用的登录号 y.

C0HLZ5	P01858	P0DPI4	P0DUS0	P84464
P84465	P0DOY5	P67857	P67858	P67859
P81826	P23210	P85003	P84071	P86168
C0HJF1	C0HJG0	P02729	P81010	P86909
P86922	B3EWE5	P0DMM6 P0DQM6 P0DQM7		
P12481	P85002	B3EWR3	P01358	P02728
P0C005	P0DKX2	P0DMM7 P0DQM9 P0DX30		
P22103	P84200	P84785	P84868	P86600
A8C8X2	B3A0L6	B3EUR5	C0HJM6	C0HLK7
P0DJC3	P0DJF4	P69208	P85444	P85870
P86942	A0A0A0MT89	B3EWS0	C0HJB6	C0HL84
C0HL88	P0C8I8	P0DQH7	P0DQH8	P0DQX4
P0DQX5	P58805	P69437	P82820	P83127

F.2 模型

我们使用 GPT-2 Large（Radford 等人，2019年）、Llama 3.2-1B、Llama 3.1-8B 和 Phi-4（14B；Abdin 等人，2024）[29]，模型来自 Hugging Face Hub（Wolf 等人，2020年）。我们使用该库[30] 和（ 等人，2023）后端，其效率估这些模型。

对于 PTB 实验，我们使用 [31] 将词元级模型转换为字节级模型，并与 $\mathtt{fptb}$组合。对于 weuseabeamsizeof K=5和剪枝阈值 0.001。对于 DNA 实验，我们训练了一个定制的 GPT-2 Small 模型[32]，使用人类 DNA 数据集[33]。该模型的词元集合为 $\mathcal{X} = \{\mathtt{A}、\mathtt{C}、\mathtt{G}、\mathtt{T}\}$，因此无需使用转换器 $\mathtt{f}_\alpha$ 将子词词元映射为字节，再与模型组合。训练参数见表 2；训练和验证指标见表 3。

F.3 参数

所有实验均采用 §C.3 中描述的剪枝启发式。对于 §7 中的实验，我们报告不同 $n_{\max} \in \{5000, 10000, 15000, 20000, 25000, 30000\}$ 取值下的结果。对于所有其他实验，我们将 $n_{\max} = \infty$。

F.4 GPU 使用情况

所有实验在§7 中均使用一块NVIDIA GeForce RTX 4090（24GB），但Phi-4除外，其需要两块。基准测试在§G.5 中使用NVIDIA GeForce RTX 3090。

[28] https://www.uniprot.org/uniprotkb?query=人类 [29] 可通过 openai/gpt2-large、meta-llama/Llama-3.2-1B、meta-llama/Llama-3.1-8B 和microsoft/phi-4 获取，网址为 https://huggingface.co。 [30] https://github.com/genlm/genlm-backend [31] https://github.com/genlm/genlm-bytes [32] https://huggingface.co/vesteinn/gpt2-dna [33] https://huggingface.co/datasets/simecek/Human_DNA_v0。

表2：在人类 DNA 上训练的 GPT-2 small 模型的训练参数。

参数	数值
学习率	0.0003
优化器	AdamW $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e{-8}$
学习率调度器	线性
预热步数	1000
训练批次（每设备）	64
评估批次（每台设备）	8
训练批次总数	256（262,144 个词元）
评估批次总数	32
训练轮数	10
Seed	42
分布式训练	多 GPU（4 个设备）
混合精度	原生自动混合精度

表 3：GPT-2small 模型在人类 DNA 上的训练与验证指标。

Step	训练轮次	训练损失	验证损失	准确率（%）
5k	0.69	1.1252	1.1206	47.45
10k	1.38	1.0835	1.0814	49.91
15k	2.07	1.0641	1.0639	51.03
20k	2.76	1.0563	1.0547	51.63
25k	3.45	1.0504	1.0486	52.04
30k	4.14	1.0439	1.0439	52.33
35k	4.84	1.0425	1.0407	52.54
40k	5.53	1.0365	1.0380	52.71
45k	6.22	1.0325	1.0361	52.84
50k	6.91	1.0322	1.0341	52.96
55k	7.60	1.0307	1.0328	53.05
60k	8.29	1.0267	1.0316	53.13
65k	8.98	1.0273	1.0306	53.20
70k	9.67	1.0270	1.0299	53.24

F.5 实验中所用转换器的详细信息

表 4 给出了所述转换器的状态数、IP-通用状态数和转移数，见§7。我们使用Pynini（Gorman，2016年）构建所有有限状态转换器。请注意，对于使用宾州树库有限状态转换器（$\mathrm{f}_{\mathrm{ptb}}$）的实验，我们实现 $p_{\mathcal{X}} \circ \mathrm{f}_{\alpha}$ 时使用 , , 从而使状态数和弧数保持不变。

GenLM bytes

表 4：状态数、IP-通用状态数和转移数。

模型	状态	IP-Univ. 状态	转移
词元到字节			
$p_{\mathsf{gpt2}} \circ \mathrm{f}_{\alpha}$	75,723	75,723	125,979
$p_{\mathsf{llama1B}} \circ \mathrm{f}_{\alpha}$	176,990	176,990	305,244
$p_{\mathsf{llama8B}} \circ \mathrm{f}_{\alpha}$	176,990	176,990	305,244
$p_{\mathsf{phi4}} \circ \mathrm{f}_{\alpha}$	115,244	115,244	215,593
词元到单词			
$p_{\mathsf{gpt2}} \circ \mathrm{f}_{\alpha} \circ \mathrm{f}_{\mathrm{ptb}}$	479	56	26,962
$p_{\mathsf{llama1B}} \circ \mathrm{f}_{\alpha} \circ \mathrm{f}_{\mathrm{ptb}}$	479	56	26,962
$p_{\mathsf{llama8B}} \circ \mathrm{f}_{\alpha} \circ \mathrm{f}_{\mathrm{ptb}}$	479	56	26,962
$p_{\mathsf{phi4}} \circ \mathrm{f}_{\alpha} \circ \mathrm{f}_{\mathrm{ptb}}$	479	56	26,962
DNA 到氨基酸			
$p_{\mathsf{dna}} \circ \mathrm{f}_{\mathrm{dna2aa}}$	21	21	84

构建 PTB 转导器。 我们通过将每条分词器规则编码为[34]一个通过插入特定分隔符来分割字符序列的有限状态转换器$\textsc{sep} \notin \mathcal{Y}$，构建 PTB 有限状态转换器。具体而言，该转换器会在标点符号和附着词边界处插入$\textsc{sep}$，并将空白字符（空格、制表符、换行符等）折叠为$\textsc{sep}$。请注意，因此得到的转导语言模型并不是一个真正定义在 PTB 词元上的分布，而是定义在字符和分隔符上的分布，这些字符和分隔符对应于 PTB 词元所具有的相同边界。这是一项务实的决定，因为 PTB 分词器可以将任何句子分词为正字法词。换言之，它将接受一个无限词汇表。这样的转换器可以动态构建，并且等价于一个具有无限多个状态的转换器。在本文中，我们只使用有限版本。图 5 给出了此类规则的一个示例：如果逗号后面不是数字，则在逗号前插入 SEP。然后，我们将

[34]请参阅 https://www.nltk.org/_modules/nltk/tokenize/treebank.html#TreebankWordTokenizer 查看完整规范。

这些有限状态转换器组合成一个单一转换器（fptb）。请注意，上下文相关规则（例如图 5 中所示的规则）会引入非 IP 通用状态。例如，状态 q_2 只接受 $x \in \mathcal{X} \setminus \{0\text{–}9,\}$。实际上，在fptb的 479 个状态中，只有 56 个是 IP-通用的。

DNA 到氨基酸转换器。§7中介绍的 DNA 到氨基酸转换器在图17中部分展示。

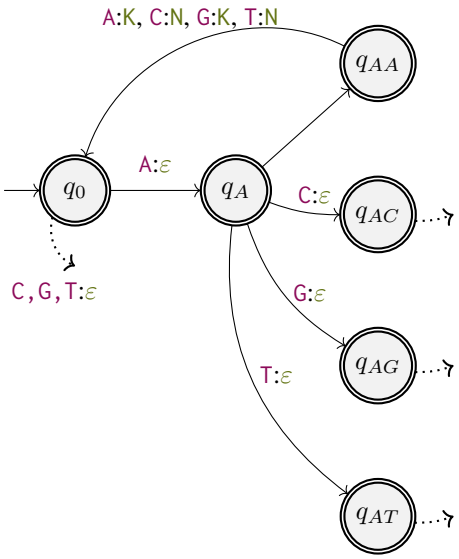


图 17：一个将 DNA 序列转换为氨基酸的有限状态转换器。每个核碱基三联体映射到 20 种不同氨基酸中的一种。我们仅展示该机器的一部分。

G 附加实验结果

此处我们使用 §C中介绍的算法，为 §7中的实验结果提供补充结果。首先，在 §G.1 中，我们将使用 $\mathtt{f}_\alpha$ 的基于转导器的方法与（Vieira 等人，2025a）进行比较，随后在 §G.2中对$\mathtt{f}_\alpha$、$\mathtt{f}^{\mathrm{ptb}}$和$\mathtt{f}_{\mathrm{dna2aa}}$给出的 D 比较。在 §G.4 中，我们报告 $\mathtt{f}_\alpha$ 的交叉熵比较，最后在 §G.5中消融分析 IP-通用状态的重要性。

G.1 与VIEIRAETAL. (2025A)的比较

Vieira 等人 (2025a) 使用束大小为 K 的束搜索算法，将词元级模型转换为字节级模型。我们通过计算 WikiText 段落上的两种方法所得分布之间的 JSD（§F），将基于转换器的方法与他们的方法进行比较。表 6 以 genlm-bytes（$K = 32$，剪枝阈值为 0.0）为参考，并改变我们的剪枝阈值 τ。随着 τ 减小，JSD 随之下降，我们的分布也逐渐趋近基线。在适中的阈值（$\tau \leq$ 1e-3）下，所有模型的 JSD 已低于 1.5e-3，证实这两种方法产生的分布非常接近。

为验证这两种方法在精确设定下计算的是相同的底层分布，我们在不进行剪枝的情况下对它们进行比较：$\tau = 0$（我们的方法不进行剪枝，genlm-bytes 使用精确推断（$K = \infty$）。表 5展示了使用 $p_{\mathsf{gpt2}} \circ \mathtt{f}_\alpha$ 对 WikiText 评估数据前 10 个字节逐位置计算的 JSD 和最大绝对差。JSD 值约为 1e-14–1e-15，最大绝对概率差不超过 3.4e-6，这证实两种方法在浮点精度范围内得出的分布相同。

表 5：精确比较：我们的方法（$\tau = 0$）与 genlm-bytes（精确）在 $p_{\mathsf{gpt2}} \circ \mathtt{f}_\alpha$ 上的比较。

Pos	Byte	JSD	最大值 $\lvert\Delta p\rvert$
0	R	3.9e-15	7.9e-7
1	o	6.5e-16	1.2e-6
2	b	3.5e-15	1.2e-6
3	e	4.4e-14	1.7e-6
4	r	4.5e-15	3.4e-6
5	t	1.8e-15	2.4e-6
6	␣	1.1e-14	1.4e-6
7	B	1.9e-15	1.8e-6
8	o	6.1e-15	1.2e-6
9	u	1.3e-14	1.5e-6

表 7 报告了基线方法本身在 $K \in \{8, 16, 32\}$ 时的 JSD 和吞吐量，展示了其方法内部的精度–速度权衡。将 K 从 32 降至 8 可大幅提高吞吐量，同时仅使 JSD 发生小幅变化，这表明基线在 $K = 32$ 时的近似误差很小。请注意，Vieira 等人 (2025a) 专为严格前缀单调变换而设计，这是一种 $\mathtt{f}_\alpha$所满足的性质。这种专门设计使我们能够采用基于trie的算法，在这类变换上实现比通用方法更好的精度–速度权衡。

表6：不同阈值下的平均詹森-香农散度（JSD）和字节/秒 τ 使用$p_{\mathcal{X}} \circ \mathrm{f}_\alpha$ 相对于来自Vieira等人（2025a）的参考分布，束大小为 K=32。95%置信区间在括号中给出。

τ	$p_{\text{gpt2}} \circ \mathrm{f}_\alpha$ 平均每字节 JSD	$p_{\text{gpt2}} \circ \mathrm{f}_\alpha$ 字节/秒	$p_{\text{llama1B}} \circ \mathrm{f}_\alpha$ 平均每字节 JSD	$p_{\text{llama1B}} \circ \mathrm{f}_\alpha$ 字节/秒
1e-1	4.0e-2 (3.7e-2, 4.2e-2)	70.95 (64.14, 79.65)	2.8e-2 (2.5e-2, 3.0e-2)	84.21 (71.77, 99.54)
3e-2	1.9e-2 (1.8e-2, 2.1e-2)	44.68 (40.59, 48.66)	1.2e-2 (1.1e-2, 1.4e-2)	72.23 (65.87, 78.10)
1e-2	6.7e-3 (5.8e-3, 7.8e-3)	25.13（23.83，26.69）	4.2e-3 (3.5e-3, 4.9e-3)	40.45 (37.60, 43.43)
3e-3	2.5e-3（1.9e-3，3.3e-3）	10.48（9.89，11.12）	1.1e-3（8.4e-4，1.3e-3）	22.09 (20.94, 23.51)
1e-3	1.4e-3 (9.3e-4, 1.9e-3)	6.21（5.89，6.59）	4.0e-4（2.9e-4，5.6e-4）	11.86（11.27，12.49）
3e-4	7.6e-5（6.0e-5，9.9e-5）	1.97（1.84，2.11）	1.3e-4（1.1e-4，1.5e-4）	5.90（5.60，6.22）
1e-4	3.7e-5 (3.0e-5, 4.7e-5)	0.65（0.61，0.71）	8.8e-5 (8.1e-5, 9.5e-5)	2.15（2.04，2.26）
3e-5	2.5e-5 (2.4e-5, 2.7e-5)	0.21（0.19，0.22）	7.8e-5 (7.3e-5, 8.3e-5)	1.20 (1.15, 1.27)
1e-5	2.5e-5 (2.3e-5, 2.7e-5)	0.19 (0.17, 0.21)	7.4e-5 (6.9e-5, 7.7e-5)	0.62（0.59，0.66）

τ	$p_{\text{llama8B}} \circ \mathrm{f}_\alpha$ 平均每字节 JSD	$p_{\text{llama8B}} \circ \mathrm{f}_\alpha$ 字节/秒	$p_{\text{phi4}} \circ \mathrm{f}_\alpha$ 平均每字节 JSD	$p_{\text{phi4}} \circ \mathrm{f}_\alpha$ 字节/秒
1e-1	2.2e-2 (2.0e-2, 2.4e-2)	79.03 (71.08, 87.28)	2.5e-2 (2.3e-2, 2.7e-2)	41.04 (36.96, 45.33)
3e-2	8.8e-3 (7.7e-3, 1.0e-2)	66.41（62.69，70.69）	9.7e-3 (8.6e-3, 1.1e-2)	36.20（33.26，38.87）
1e-2	3.5e-3（2.9e-3，4.2e-3）	45.47 (43.10, 47.91)	4.6e-3 (3.8e-3, 5.4e-3)	22.85（21.14，24.77）
3e-3	1.3e-3（9.3e-4，1.8e-3）	26.71 (25.11, 28.34)	1.8e-3 (1.4e-3, 2.4e-3)	14.80（13.70，15.83）
1e-3	5.1e-4 (2.9e-4, 8.4e-4)	15.73 (14.88，16.60)	8.5e-4 (5.0e-4, 1.3e-3)	8.75 (8.30, 9.27)
3e-4	1.3e-4 (1.1e-4, 1.5e-4)	9.08（8.68，9.57）	1.9e-4 (1.4e-4, 2.5e-4)	4.87 (4.60, 5.16)
1e-4	9.1e-5 (8.0e-5, 1.0e-4)	4.03（3.84，4.23）	1.0e-4 (8.8e-5, 1.2e-4)	2.19 (2.07, 2.33)
3e-5	7.5e-5（6.8e-5，8.3e-5）	2.38（2.27，2.49）	7.6e-5 (6.8e-5, 8.4e-5)	1.30 (1.23, 1.37)
1e-5	7.3e-5 (6.6e-5, 8.1e-5)	1.40（1.33，1.47）	7.9e-5（7.2e-5，8.8e-5）	0.74（0.70，0.79）

表7：Vieira 等人（2025a）在束宽$K \in \{8, 16, 32\}$下的吞吐量和詹森-香农散度（JSD） $K = 32$。95%置信区间以括号表示。

K	$p_{\text{gpt2}} \circ \mathrm{f}_\alpha$ JSD 与 $K=32$ 的对比	$p_{\text{gpt2}} \circ \mathrm{f}_\alpha$ 字节/秒	$p_{\text{llama1B}} \circ \mathrm{f}_\alpha$ JSD 与 $K=32$	$p_{\text{llama1B}} \circ \mathrm{f}_\alpha$ 字节/秒
8	3.0e-5 (2.5e-5, 3.9e-5)	37.81 (37.59, 37.98)	7.1e-5（6.7e-5，7.6e-5）	46.94 (45.69, 47.97)
16	2.4e-5 (2.3e-5, 2.7e-5)	28.83 (28.53, 29.13)	7.1e-5 (6.7e-5, 7.5e-5)	28.76（28.42，29.08）
32	(不适用)	18.12（17.87，18.36）	(不适用)	5.34（4.21，7.19）

K	$p_{\text{llama8B}} \circ \mathrm{f}_\alpha$ JSD 与 $K=32$	$p_{\text{llama8B}} \circ \mathrm{f}_\alpha$ 字节/秒	$p_{\text{phi4}} \circ \mathrm{f}_\alpha$ JSD 与 $K=32$	$p_{\text{phi4}} \circ \mathrm{f}_\alpha$ 字节/秒
8	6.8e-5（6.3e-5，7.4e-5）	18.30（18.17，18.43）	4.6e-5 (4.3e-5, 4.9e-5)	11.59（11.29，11.80）
16	6.9e-5 (6.2e-5, 7.8e-5)	10.55 (10.45, 10.65)	5.3e-5 (4.9e-5, 5.6e-5)	7.90（7.84，7.96）
32	（不适用）	5.85 (5.79, 5.91)	（不适用）	4.51（4.44，4.56）

G.2 詹森–香农散度

我们对 §C 中给出的算法进行基准测试，评估其在概率质量剪枝中采用较高剪枝阈值 τ 时对精确分布的近似效果；该剪枝方法见 §C.3。在三种设置——词元到字节、PTB 分词和 DNA 到氨基酸——中，随着 τ 降低，JSD 也随之降低，但吞吐量（字节/秒）会下降。

表 8 报告了所有四个模型的词元到字节结果（$p_{\mathcal{X}} \circ \mathrm{f}_\alpha$）。JSD 从 $\tau =$ 1e-1 降低了两到三个数量级，至 $\tau =$ 3e-5；阈值最严格时，所有模型的 JSD 均低于 1e-4。

表 9 给出了 PTB 分词的相应结果（$p_{\mathcal{X}} \circ \mathrm{f}_\alpha \circ \mathrm{f}_{\text{ptb}}$），补充了图 6（右）。虽然 f_{ptb} 的状态数少于词元到字节转换器，但其 479 个状态中只有 56 个是 IP-通用的，因此较低的阈值在计算上代价高昂；所以参考值设为 $\tau =$1e-4（对于 p_{phi4}，则为 3e-4），而非 1e-5。尽管如此，JSD 收敛很快，在 $\tau =$ 3e-4 时，对于 p_{gpt2}、p_{llama1B}和 p_{llama8B}，JSD 已达到 1e-5 量级；而 p_{phi4} 的收敛速度较慢。

表 10 报告了 DNA 到氨基酸的结果（$p_{\mathsf{dna}} \circ \mathrm{f}_{\mathrm{dna2aa}}$），其中我们还改变了候选集上限 n_{max}，以缓解三对一核苷酸到氨基酸映射固有的组合爆炸。更严格的阈值通常会降低 JSD。然而，τ 与 n_{max} 之间的相互作用并不那么一致，因为评估分布和参考分布都取决于上限。

未完成的 run。如 §G.3 所示，在严格阈值下，分解大小会迅速增长——在 f_α 的 $\tau = 1\text{e-}5$ 阈值下，平均 $|Q|$ 超过 35,000。对于某些模型–阈值组合，这种增长导致无法计算全部 10 个评估段落中每个位置的完整分布。具体而言，表 8 中的 $p_{\mathsf{gpt2}} \circ \mathrm{f}_\alpha$ 在 $\tau = 1\text{e-}5$ 阈值下仅完成了 10 个段落中的 7 个。在表 9 中，$p_{\mathsf{phi4}} \circ \mathrm{f}_\alpha \circ \mathrm{f}^{\mathrm{ptb}}$ 在其参考阈值 $\tau = 3\text{e-}4$ 下完成了 10 个段落中的 7 个，而 $p_{\mathsf{llama1B}} \circ \mathrm{f}_\alpha \circ \mathrm{f}^{\mathrm{ptb}}$ 在 $\tau = 1\text{e-}4$ 下完成了 10 个中的 9 个。此外，由于发生了无法恢复的回溯失败（图 10），$p_{\mathsf{phi4}} \circ \mathrm{f}_\alpha \circ \mathrm{f}^{\mathrm{ptb}}$ 在 $\tau = 1\text{e-}1$ 下仅完成了 10 个段落中的 9 个。在这些情况下，JSD 和吞吐量均基于该模型–转换器组合在所有阈值下共同完成的段落计算。

表 8：不同阈值下，使用 τ $p_{\mathcal{X}} \circ \mathrm{f}_\alpha$ 与参考分布（$\tau = 1\text{e-}5$）进行比较时的平均詹森-香农散度（JSD）和字节/秒。95% 置信区间列于括号内。

	$p_{\mathsf{gpt2}} \circ \mathrm{f}_\alpha$		$p_{\mathsf{llama1B}} \circ \mathrm{f}_\alpha$	
τ	平均每字节 JSD	字节/秒	平均每字节 JSD	字节/秒
1e-1	3.8e-2 (3.4e-2, 4.1e-2)	76.50（66.94，89.01）	2.8e-2 (2.6e-2, 3.0e-2)	84.21 (72.49, 99.24)
3e-2	1.7e-2（1.5e-2，1.9e-2）	45.61 (39.82, 51.46)	1.2e-2（1.1e-2，1.4e-2）	72.23 (66.25, 78.94)
1e-2	5.9e-3 (4.8e-3, 7.1e-3)	25.71（24.08，27.89）	4.2e-3 (3.5e-3, 5.0e-3)	40.45（37.65，43.43）
3e-3	2.1e-3 (1.4e-3, 2.9e-3)	11.22（10.44，12.07）	1.1e-3 (8.5e-4, 1.3e-3)	22.09（20.90，23.34）
1e-3	1.5e-3 (9.1e-4, 2.3e-3)	6.49 (6.02, 6.99)	4.0e-4 (2.8e-4, 5.4e-4)	11.86 (11.25, 12.54)
3e-4	5.5e-5 (4.7e-5, 6.7e-5)	2.28 (2.09, 2.48)	1.2e-4 (1.1e-4，1.4e-4)	5.90 (5.59, 6.23)
1e-4	3.6e-5 (2.9e-5, 4.8e-5)	1.00（0.92，1.09）	7.4e-5 (6.9e-5, 8.0e-5)	2.15 (2.03, 2.26)
3e-5	2.4e-5 (2.2e-5, 2.7e-5)	0.44 (0.40, 0.49)	5.2e-5 (4.9e-5, 5.6e-5)	1.20（1.14，1.26）
1e-5	(不适用)	0.19 (0.17, 0.21)	(不适用)	0.62 (0.59, 0.65)

	$p_{\mathsf{llama8B}} \circ \mathrm{f}_\alpha$		$p_{\mathsf{phi4}} \circ \mathrm{f}_\alpha$	
τ	平均每字节 JSD	字节/秒	平均每字节 JSD	字节/秒
1e-1	2.3e-2 (2.1e-2, 2.5e-2)	79.03 (71.62, 87.24)	2.5e-2 (2.3e-2, 2.7e-2)	41.04 (36.75, 45.30)
3e-2	8.9e-3（7.8e-3，1.0e-2）	66.41 (61.96, 70.51)	9.9e-3（8.7e-3，1.1e-2）	36.20 (33.36, 38.85)
1e-2	3.5e-3（2.9e-3，4.4e-3）	45.47 (43.16, 48.08)	4.7e-3 (3.8e-3, 5.6e-3)	22.85（21.06，24.55）
3e-3	1.3e-3 (9.4e-4, 1.8e-3)	26.71（25.17，28.24）	1.8e-3 (1.3e-3, 2.4e-3)	14.80 (13.74, 15.78)
1e-3	5.2e-4 (3.0e-4, 8.4e-4)	15.73（14.89，16.61）	8.3e-4 (4.8e-4, 1.2e-3)	8.75 (8.30, 9.27)
3e-4	1.2e-4 (1.1e-4，1.4e-4)	9.08 (8.64, 9.60)	1.9e-4 (1.1e-4, 2.9e-4)	4.87（4.61，5.19）
1e-4	8.3e-5 (7.3e-5, 9.6e-5)	4.03（3.83，4.26）	7.4e-5 (6.1e-5, 9.2e-5)	2.19 (2.05, 2.33)
3e-5	5.9e-5 (5.3e-5, 6.6e-5)	2.38 (2.27，2.49)	3.7e-5 (3.4e-5, 4.0e-5)	1.30（1.23，1.37）
1e-5	(不适用)	1.40（1.34，1.47）	(不适用)	0.74（0.70，0.79）

表 9：采用 τ 的$p_{\mathcal{X}} \circ \mathrm{f}_{\alpha} \circ \mathrm{f}^{\mathrm{ptb}}$ 时，相对于参考分布（$\tau = $ 1e-4；对于 p_{phi4}，3e-4）。平均詹森-香农散度（JSD）和字节/秒（95% 置信区间见括号内。

τ	$p_{\mathsf{gpt2}} \circ \mathrm{f}_{\alpha} \circ \mathrm{f}^{\mathrm{ptb}}$ 平均每字节 JSD	字节/秒	$p_{\mathsf{llama1B}} \circ \mathrm{f}_{\alpha} \circ \mathrm{f}^{\mathrm{ptb}}$ 平均每字节 JSD	字节/秒
1e-1	3.9e-3（3.3e-3，4.7e-3）	26.65 (25.79, 27.48)	3.0e-3 (2.6e-3, 3.5e-3)	30.44（28.31，32.19）
3e-2	1.4e-3 (1.2e-3, 1.5e-3)	17.03 (16.42, 17.66)	1.1e-3（9.3e-4，1.3e-3）	17.45（16.67，18.27）
1e-2	5.7e-4 (5.0e-4, 6.7e-4)	10.79（10.23，11.36）	3.7e-4（3.4e-4，4.0e-4）	10.54（10.03，11.09）
3e-3	1.7e-4 (1.5e-4, 1.9e-4)	4.52 (4.29, 4.79)	1.3e-4 (1.2e-4, 1.4e-4)	8.30（7.98，8.67）
1e-3	5.5e-5 (5.1e-5, 6.1e-5)	2.01（1.88，2.15）	6.1e-5（5.8e-5，6.5e-5）	4.13（3.91，4.36）
3e-4	2.0e-5 (1.7e-5, 2.4e-5)	0.81（0.77，0.85）	3.3e-5（3.1e-5，3.5e-5）	1.67（1.59，1.76）
1e-4	(不适用)	0.20（0.16，0.25）	(不适用)	0.70 (0.66，0.74)

τ	$p_{\mathsf{llama8B}} \circ \mathrm{f}_{\alpha} \circ \mathrm{f}^{\mathrm{ptb}}$ 平均每字节 JSD	字节/秒	$p_{\mathsf{phi4}} \circ \mathrm{f}_{\alpha} \circ \mathrm{f}^{\mathrm{ptb}}$ 平均每字节 JSD	字节/秒
1e-1	2.5e-3 (2.1e-3, 3.0e-3)	19.25（18.79，19.76）	4.6e-3（3.2e-3，6.5e-3）	10.07（9.53，10.67）
3e-2	1.0e-3（8.6e-4，1.2e-3）	13.38（12.99，13.83）	3.8e-3（2.6e-3，5.1e-3）	5.87 (5.37, 6.40)
1e-2	6.0e-4（4.7e-4，7.8e-4）	9.49（9.16，9.88）	3.2e-3 (2.2e-3, 4.4e-3)	2.94 (2.59, 3.35)
3e-3	1.1e-4 (1.0e-4, 1.2e-4)	5.90（5.62, 6.20）	4.2e-4 (3.0e-4, 5.8e-4)	2.95 (2.68, 3.26)
1e-3	5.1e-5（4.7e-5，5.5e-5）	3.29（3.13，3.45）	3.6e-4（2.5e-4，4.8e-4）	1.54（1.37，1.74）
3e-4	2.3e-5 (2.1e-5, 2.5e-5)	1.42（1.35，1.49）	(不适用)	0.48（0.43，0.54）
1e-4	(不适用)	0.56（0.53，0.59）	（不适用）	（不适用）

表10：不同阈值下的平均詹森-香农散度（JSD）和字节/秒 τ 以及一个参考值（τ =1e-6），使用 $p_{\mathsf{dna}} \circ \mathsf{f}_{\mathrm{dna2aa}}$。95% 置信区间在括号中给出。我们将候选集大小（n_{max}）加以限制，以缓解随序列长度增加而产生的组合爆炸。

τ	$p_{\mathsf{dna}} \circ \mathsf{f}_{\mathrm{dna2aa}}$ (n_{max} = 5000) 平均每字节 JSD	字节/秒	$p_{\mathsf{dna}} \circ \mathsf{f}_{\mathrm{dna2aa}}$ (n_{max} = 10000) 平均每字节 JSD	字节/秒
1e-1	2.8e-2(2.4e-2,3.2e-2)	9.55 (8.44, 10.90)	2.8e-2 (2.5e-2, 3.3e-2)	9.54 (8.45, 10.82)
3e-2	3.2e-3 (2.8e-3, 3.6e-3)	2.90 (2.57, 3.26)	3.3e-3（2.9e-3，3.7e-3）	2.03（1.74，2.39）
1e-2	6.6e-4（4.4e-4，9.0e-4）	2.20 (1.99, 2.47)	7.4e-4 (5.0e-4, 1.0e-3)	1.42 (1.25, 1.63)
3e-3	1.6e-4（7.8e-5，2.7e-4）	1.89（1.72，2.11）	2.7e-4（1.1e-4，4.9e-4）	1.16 (1.03, 1.31)
1e-3	7.1e-5 (1.7e-5, 1.6e-4)	1.79（1.62，1.97）	6.6e-5 (1.7e-5, 1.5e-4)	1.08（0.97，1.22）
3e-4	4.9e-5 (9.1e-6, 1.1e-4)	1.73（1.58，1.93）	3.5e-5 (6.7e-6, 7.9e-5)	1.04（0.93，1.17）
1e-4	5.8e-5 (1.1e-5, 1.2e-4)	1.72 (1.56, 1.91)	2.1e-5 (9.9e-7, 7.9e-5)	1.02 (0.92, 1.16)
3e-5	3.0e-5 (7.9e-7, 7.9e-5)	1.71（1.55，1.89）	2.1e-5 (6.6e-7, 6.2e-5)	1.02（0.92，1.14）
1e-5	2.2e-5 (1.0e-6, 5.4e-5)	1.71（1.56，1.89）	3.9e-6 (5.1e-7, 8.9e-6)	1.02（0.91，1.15）
3e-6	1.0e-5 (2.3e-7, 2.9e-5)	1.70（1.55，1.89）	1.2e-5 (6.1e-7, 3.2e-5)	1.01（0.91，1.12）
1e-6	(不适用)	1.71（1.55，1.90）	(不适用)	1.02（0.92，1.14）

τ	$p_{\mathsf{dna}} \circ \mathsf{f}_{\mathrm{dna2aa}}$ (n_{max} = 15000) 平均每字节 JSD	字节/秒	$p_{\mathsf{dna}} \circ \mathsf{f}_{\mathrm{dna2aa}}$ (n_{max} = 20000) 平均每字节 JSD	字节/秒
1e-1	2.9e-2(2.5e-2,3.3e-2)	9.52 (8.46, 10.82)	2.9e-2 (2.5e-2, 3.3e-2)	9.35 (8.21, 10.65)
3e-2	3.2e-3（2.8e-3，3.5e-3）	1.63（1.37，1.94）	3.2e-3 (2.8e-3, 3.7e-3)	1.41（1.18，1.71）
1e-2	6.8e-4 (4.5e-4, 9.8e-4)	1.10 (0.96, 1.27)	8.3e-4（5.0e-4，1.3e-3）	0.91 (0.78, 1.07)
3e-3	2.2e-4（8.2e-5，4.2e-4）	0.88（0.77，1.00）	2.0e-4（7.6e-5，3.9e-4）	0.71（0.62，0.83）
1e-3	5.0e-5（8.5e-6，1.3e-4）	0.80（0.71，0.91）	6.6e-5（1.7e-5，1.5e-4）	0.64（0.56，0.73）
3e-4	2.8e-5 (3.6e-6, 7.2e-5)	0.77 (0.68, 0.88)	3.0e-5 (3.9e-6, 7.9e-5)	0.61 (0.55, 0.71)
1e-4	2.4e-5 (1.1e-6, 7.0e-5)	0.75（0.66，0.84）	6.0e-5 (1.8e-6, 1.6e-4)	0.60 (0.54, 0.69)
3e-5	6.0e-5 (1.0e-6, 1.6e-4)	0.74（0.66，0.84）	1.8e-4 (5.0e-5, 3.7e-4)	0.60 (0.53, 0.68)
1e-5	3.0e-7 (7.1e-9, 8.3e-7)	0.74 (0.66, 0.84)	7.8e-7 (5.3e-8, 2.1e-6)	0.59（0.53，0.68）
3e-6	9.3e-7 (1.1e-8, 2.7e-6)	0.73（0.65，0.83）	7.3e-7 (8.9e-9, 2.1e-6)	0.59（0.53，0.67）
1e-6	(不适用)	0.74（0.65，0.84）	(不适用)	0.60 (0.53, 0.69)

τ	$p_{\mathsf{dna}} \circ \mathsf{f}_{\mathrm{dna2aa}}$ (n_{max} = 25000) 平均每字节 JSD	字节/秒	$p_{\mathsf{dna}} \circ \mathsf{f}_{\mathrm{dna2aa}}$ (n_{max} = 30000) 平均每字节 JSD	字节/秒
1e-1	2.9e-2(2.5e-2,3.3e-2)	9.50 (8.31, 10.78)	2.8e-2 (2.5e-2, 3.3e-2)	9.52 (8.43, 10.99)
3e-2	3.3e-3 (2.9e-3, 3.7e-3)	1.28 (1.06, 1.60)	3.3e-3 (2.9e-3, 3.8e-3)	1.18 (0.96, 1.45)
1e-2	7.9e-4 (4.9e-4, 1.3e-3)	0.81（0.70，0.95）	8.2e-4（4.9e-4，1.2e-3）	0.72（0.61，0.86）
3e-3	2.3e-4 (8.1e-5, 4.7e-4)	0.62 (0.54, 0.72)	2.5e-4 (7.9e-5, 4.8e-4)	0.55（0.47，0.64）
1e-3	5.9e-5 (1.1e-5, 1.4e-4)	0.56（0.49，0.65）	6.0e-5 (1.1e-5, 1.4e-4)	0.49（0.42，0.57）
3e-4	2.6e-5 (3.5e-6, 6.8e-5)	0.53（0.46，0.61）	3.4e-5 (4.9e-6, 8.5e-5)	0.46 (0.41, 0.53)
1e-4	2.4e-5 (8.2e-7, 6.9e-5)	0.52（0.46，0.59）	2.4e-5 (1.0e-6, 8.0e-5)	0.45（0.39，0.51）
3e-5	7.2e-5 (1.9e-5, 1.5e-4)	0.51（0.45，0.59）	2.4e-5 (5.6e-7, 7.1e-5)	0.45 (0.39，0.52)
1e-5	1.9e-6 (5.6e-8, 4.6e-6)	0.51 (0.45, 0.59)	1.7e-7 (5.7e-9, 3.8e-7)	0.44（0.39，0.51）
3e-6	1.3e-6 (1.2e-8, 3.3e-6)	0.51 (0.45, 0.58)	1.6e-6（1.7e-8，3.7e-6）	0.44 (0.38, 0.51)
1e-6	(不适用)	0.51（0.45，0.59）	(不适用)	0.44（0.39，0.51）

G.3 分解大小

表 11 和图 18 展示了商与余项大小如何随 τ 的减小而增长，这解释了 JSD 实验中观察到的吞吐量下降。

对于全 IP-通用的词元到字节转换器和 DNA 转导器，每个位置的余项均为空（$|\mathsf{R}| = 0$），因此仅报告商大小。在词元到字节设置中，平均商大小从 τ = 1e-1 时的 62 增至 τ = 1e-5 时的 35,000 以上，这直接解释了严格阈值下的吞吐量下降。DNA 转导器则因 $n_{\mathsf{max}} = 5000$而较早达到饱和：平均 $|\mathsf{Q}|$到 τ = 3e-5 时达到 465，此后保持平稳。

对于 $\mathsf{f}_{\mathrm{ptb}}$，它具有非 IP 通用状态，报告了 $|\mathsf{Q}|$ 和 $|\mathsf{R}|$。$|\mathsf{Q}|$ 会随着 τ收紧而稳步增长，而 $|\mathsf{R}|$ 则会随着更多候选项进入商而先减少，随后随着对非 IP 通用状态的扩展循环发现更多余项元素而增长。存在非平凡余项，是它与全IP通用转换器之间的关键结构差异：每个

余项元素都需要完整字符串概率 $p_{\mathcal{X}}(\boldsymbol{x})$，而不只是前缀概率，因此每个元素的计算成本更高。

表 11：三个转换器在不同剪枝阈值下，各序列位置上的平均商大小和最大商大小 $|Q|$ τ。结果是在 WikiText 第1段上计算的（$p_{\text{gpt2}} \circ \mathtt{f}_{\alpha}$为 833 字节，$p_{\text{gpt2}} \circ \mathtt{f}_{\alpha} \circ \mathtt{f}^{\text{ptb}}$为 850 字节），以及在测试集中最长的蛋白质上计算的（P83127，$p_{\text{dna}} \circ \mathtt{f}_{\text{dna2aa}}$为 12 个氨基酸）。$\mathtt{f}_{\alpha}$ 和$\mathtt{f}_{\text{dna2aa}}$ 转换器具有全通用状态（$|R| = 0$处处如此）。$\mathtt{f}^{\text{ptb}}$ 转换器具有非通用状态，因此我们还报告余项大小 $|R|$。

(a) $p_{\text{gpt2}} \circ \mathtt{f}_{\alpha}$

τ	平均值 $\lvert Q\rvert$	最大值 $\lvert Q\rvert$
1e-1	62	2,238
3e-2	200	5,736
1e-2	439	11,998
3e-3	873	18,761
1e-3	1,536	25,341
3e-4	3,161	65,536
1e-4	7,800	131,072
3e-5	17,232	524,288
1e-5	35,768	2,097,152

(b) $p_{\text{dna}} \circ \mathtt{f}_{\text{dna2aa}}$

τ	均值 $\lvert Q\rvert$	最大 $\lvert Q\rvert$
1e-1	104	344
3e-2	288	1,274
1e-2	346	1,329
3e-3	439	2,011
1e-3	454	2,027
3e-4	462	2,044
1e-4	463	2,044
3e-5	465	2,046
1e-5	465	2,046
3e-6	465	2,046

(c) $p_{\text{gpt2}} \circ \mathtt{f}_{\alpha} \circ \mathtt{f}^{\text{ptb}}$

τ	均值 $\lvert Q\rvert$	最大 $\lvert Q\rvert$	平均值 $\lvert R\rvert$	最大 $\lvert R\rvert$
1e-1	1.3	31	1.0	113
3e-2	1.9	57	0.38	113
1e-2	3.6	142	0.48	25
3e-3	7.3	353	1.2	53
1e-3	16	686	3.5	129
3e-4	47	1,807	11.4	408
1e-4	128	5,101	33.9	1,195

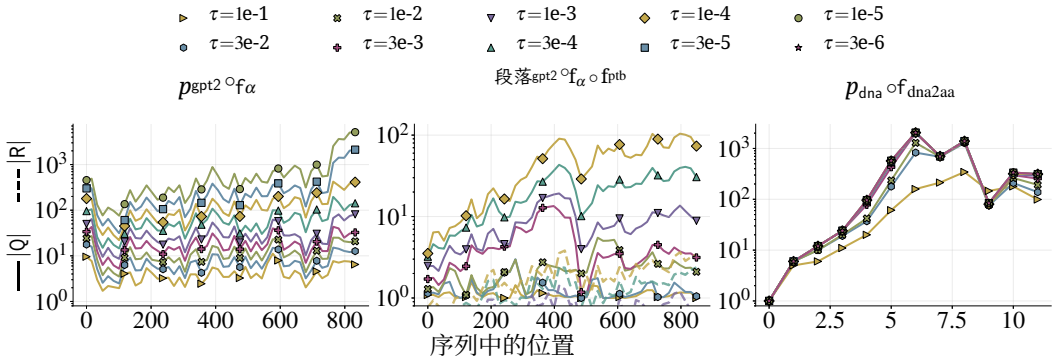


图18：在不同剪枝阈值 τ 下，三种转换器组合的商大小 $|Q|$（实线）和余项大小 $|R|$（虚线）随序列位置的变化。左：WikiText 第1段（833 字节）上的 $p_{\text{gpt2}} \circ \mathtt{f}_{\alpha}$。中：第1段上的 $p_{\text{gpt2}} \circ \mathtt{f}_{\alpha} \circ \mathtt{f}^{\text{ptb}}$（转导后为 850 字节）；虚线表示 $|R|$，其值非零，因为 $\mathtt{f}^{\text{ptb}}$ 包含非 IP 通用状态。右：测试集中最长的蛋白质（P83127，12 个氨基酸）上的 $p_{\text{dna}} \circ \mathtt{f}_{\text{dna2aa}}$。对于 $\mathtt{f}_{\alpha}$ 和 $\mathtt{f}_{\text{dna2aa}}$，所有有限状态转导器状态都是 IP-通用的，因此 $|R| = 0$处处为零。$\mathtt{f}_{\text{dna2aa}}$ 面板使用 $n_{\max} = 5{,}000$ 的候选集上限，这会限制较低阈值下的商大小，并使后续位置不再呈单调增长。

G.4 交叉熵

交叉熵衡量转导模型为留出文本分配概率的效果；与 JSD 不同，它只需评估 $\overrightarrow{p_{\mathcal{Y}}}(y \mid \boldsymbol{y})$ 中观测到的下一符号，而不必计算完整的下一符号分布 $\overrightarrow{p_{\mathcal{Y}}}(\cdot \mid \boldsymbol{y})$。表 12 报告了（$p_{\mathcal{X}} \circ \mathtt{f}_\alpha$）的交叉熵（以纳特和每字节比特数为单位）及吞吐量。由于每个位置只需计算一个概率，其吞吐量高于上文 JSD 表中的吞吐量。随着 τ 降低，交叉熵会迅速收敛：对于全部四个模型，最严格的阈值所得数值几乎相同，这证实适度剪枝就足以实现准确的序列评分。

表12：不同阈值下的交叉熵和吞吐量 τ 使用 $p_{\mathcal{X}} \circ \mathtt{f}_\alpha$. 95% 置信区间以括号中的数值表示。$p_{\text{gpt2}} \circ \mathtt{f}_\alpha$ 当 $\tau = $ 1e-5 时，计算基于 6 个段落；其他所有配置均使用 10 个段落。

τ	字节/秒		位/字节		交叉熵	
	Mean	95% 置信区间	Mean	95% 置信区间	Mean	95% 置信区间
			$p_{\text{gpt2}} \circ \mathtt{f}_\alpha$			
1e-1	82.61	(73.33, 92.34)	1.2434	(1.1884, 1.2936)	0.8619	(0.8237, 0.8966)
3e-2	49.10	(44.48, 54.24)	1.1466	(1.0978, 1.1949)	0.7948	(0.7609, 0.8282)
1e-2	28.46	(26.73, 30.33)	1.0652	(1.0199, 1.1068)	0.7383	(0.7070, 0.7672)
3e-3	14.20	(13.47, 14.98)	1.0385	(0.9956, 1.0806)	0.7199	(0.6901, 0.7490)
1e-3	6.46	(6.08, 6.84)	1.0293	(0.9909, 1.0664)	0.7134	(0.6868, 0.7392)
3e-4	2.18	(2.04, 2.33)	1.0220	(0.9831, 1.0604)	0.7084	(0.6815, 0.7350)
1e-4	0.75	(0.70, 0.80)	1.0220	(0.9823, 1.0610)	0.7084	(0.6809, 0.7355)
3e-5	0.35	(0.33, 0.39)	1.0200	(0.9781, 1.0633)	0.7070	(0.6779, 0.7370)
1e-5	0.34	(0.30, 0.38)	1.0025	(0.9461, 1.0550)	0.6949	(0.6558, 0.7313)
			$p_{\text{llama1B}} \circ \mathtt{f}_\alpha$			
1e-1	129.22	(112.82, 146.19)	0.9922	(0.9467, 1.0405)	0.6877	(0.6562, 0.7212)
3e-2	90.74	(82.63, 99.33)	0.9128	(0.8682, 0.9542)	0.6327	(0.6018, 0.6614)
1e-2	47.81	(44.04, 51.52)	0.8626	(0.8252, 0.9027)	0.5979	(0.5720, 0.6257)
3e-3	26.15	(24.71, 27.73)	0.8402	(0.8037, 0.8790)	0.5823	(0.5571, 0.6093)
1e-3	14.89	(14.18, 15.69)	0.8364	(0.8001, 0.8725)	0.5797	(0.5546, 0.6048)
3e-4	6.80	(6.46, 7.18)	0.8358	(0.7999, 0.8724)	0.5793	(0.5545, 0.6047)
1e-4	2.62	(2.49, 2.77)	0.8362	(0.8018, 0.8699)	0.5796	(0.5558, 0.6030)
3e-5	1.42	(1.35, 1.49)	0.8361	(0.8002, 0.8717)	0.5796	(0.5547, 0.6042)
1e-5	0.73	(0.69, 0.77)	0.8360	(0.8002, 0.8717)	0.5795	(0.5546, 0.6042)
			$p_{\text{llama8B}} \circ \mathtt{f}_\alpha$			
1e-1	88.65	(79.18, 99.61)	0.8176	(0.7757, 0.8661)	0.5667	(0.5377, 0.6003)
3e-2	76.39	(71.27, 81.35)	0.7392	(0.7007, 0.7759)	0.5124	(0.4857, 0.5378)
1e-2	52.05	(49.41, 54.93)	0.7153	(0.6778, 0.7507)	0.4958	(0.4698, 0.5203)
3e-3	28.81	(27.18, 30.44)	0.6959	(0.6627, 0.7282)	0.4824	(0.4594, 0.5047)
1e-3	16.86	(16.01, 17.82)	0.6913	(0.6584, 0.7251)	0.4791	(0.4564, 0.5026)
3e-4	9.73	(9.24, 10.25)	0.6870	(0.6513, 0.7179)	0.4762	(0.4515, 0.4976)
1e-4	4.35	(4.15, 4.58)	0.6870	(0.6557, 0.7221)	0.4762	(0.4545, 0.5005)
3e-5	2.60	(2.47, 2.73)	0.6869	(0.6533, 0.7208)	0.4761	(0.4528, 0.4996)
1e-5	1.57	(1.49, 1.66)	0.6869	(0.6552, 0.7180)	0.4761	(0.4541, 0.4977)
			$p_{\text{phi4}} \circ \mathtt{f}_\alpha$			
1e-1	42.39	(37.95, 47.19)	0.8714	(0.8217, 0.9208)	0.6040	(0.5695, 0.6382)
3e-2	34.48	(31.73, 37.33)	0.7748	(0.7317, 0.8180)	0.5371	(0.5072, 0.5670)
1e-2	24.89	(23.05, 26.59)	0.7547	(0.7165, 0.7959)	0.5231	(0.4966, 0.5517)
3e-3	16.55	(15.34, 17.92)	0.7301	(0.6959, 0.7690)	0.5061	(0.4824, 0.5330)
1e-3	9.91	(9.41, 10.52)	0.7241	(0.6882, 0.7595)	0.5019	(0.4770, 0.5264)
3e-4	5.81	(5.51, 6.14)	0.7170	(0.6803, 0.7509)	0.4970	(0.4716, 0.5205)
1e-4	2.55	(2.41, 2.70)	0.7163	(0.6824, 0.7493)	0.4965	(0.4730, 0.5194)
3e-5	1.54	(1.46, 1.62)	0.7162	(0.6793, 0.7490)	0.4965	(0.4709, 0.5192)
1e-5	0.86	(0.81, 0.91)	0.7162	(0.6828, 0.7502)	0.4964	(0.4733, 0.5200)

G.5 对计算捷径进行基准测试

我们对 §4 中所述前缀分解内在的计算捷径进行基准测试。该分解使我们能够将预覆盖分解为余项和商代表元。我们通过在转换器中随机选取 n 个 IP-通用状态（见 §4），并在算法中将它们视为非IP通用状态（见 §5），来生成次优分解。这样会逐步缩小商的规模，并相应地增大余项。表 13 展示了在 `wikitext-2-raw-v1` 数据集（Merity 等人，2017）前 5 个段落的前 256 个字节上，原始分布与修改后分布之间的平均 JSD。对于每个 n 值，我们重复三次采样新的非通用状态，并报告 JSD 均值。

这些分布迅速发散；在转换约 15–20% 的 IP-通用状态后，算法会反复遇到死胡同。这个数值在不同 run 之间有所变化；如图 19所示，每次重复实验都呈现出独特的阶跃函数：JSD 在转换某个特定的高连通性状态之前一直保持较低水平，随后跃升 2–3 个数量级并趋于平台。在 PTB 转导器中，一个 IP-通用状态（状态 193）充当路由枢纽，其包含跨越 245 个输出符号的 274 条弧。当该状态被转换后，包含它的前沿会从商 `Q` 移动到余项 `R`，导致扩展循环需要使用回退评分（见图 10）。不同 run 会因排列顺序不同而在不同数值处遇到该枢纽，从而产生阶梯状模式。这表明，IP-通用状态具有层级重要性结构：分布质量由少量高连通性枢纽状态决定，而大多数状态都可以在几乎不产生影响的情况下完成转换（JSD $\leq$ 1e-4）。

表 13：PTB 转换器（τ =1e-3）在随机将 n 个 IP-通用状态转换为非IP通用状态后的平均詹森–香农散度（JSD）。括号中给出 95%置信区间。JSD 是相对于未修改的转换器（$n = 0$）计算的。

转换后的状态 (n)	$p_{\text{llama1B}} \circ \mathrm{f}_\alpha \circ \mathrm{f}^{\text{ptb}}$	$p_{\text{llama8B}} \circ \mathrm{f}_\alpha \circ \mathrm{f}^{\text{ptb}}$	$p_{\text{phi4}} \circ \mathrm{f}_\alpha \circ \mathrm{f}^{\text{ptb}}$
0	(不适用)	(不适用)	不适用
2	3.9e-5（3.5e-5，4.3e-5）	1.3e-4 (7.4e-5, 2.1e-4)	6.6e-5（3.1e-5，1.1e-4）
5	9.5e-4（7.5e-4，1.2e-3）	9.0e-4（7.5e-4，1.1e-3）	7.8e-4（6.4e-4，9.4e-4）
8	9.8e-4 (8.0e-4, 1.2e-3)	9.2e-4（7.7e-4，1.1e-3）	8.1e-4 (6.6e-4, 9.6e-4)
11	3.0e-2 (2.6e-2, 3.3e-2)	2.8e-2 (2.5e-2, 3.1e-2)	2.6e-2 (2.3e-2, 2.9e-2)
14	2.9e-2 (2.6e-2, 3.3e-2)	2.8e-2 (2.5e-2, 3.1e-2)	2.6e-2 (2.3e-2, 2.9e-2)
16	2.9e-2 (2.6e-2, 3.2e-2)	2.8e-2（2.5e-2，3.2e-2）	2.6e-2 (2.3e-2, 2.9e-2)
19	2.9e-2 (2.6e-2, 3.3e-2)	2.8e-2 (2.5e-2, 3.2e-2)	2.6e-2 (2.3e-2, 2.9e-2)

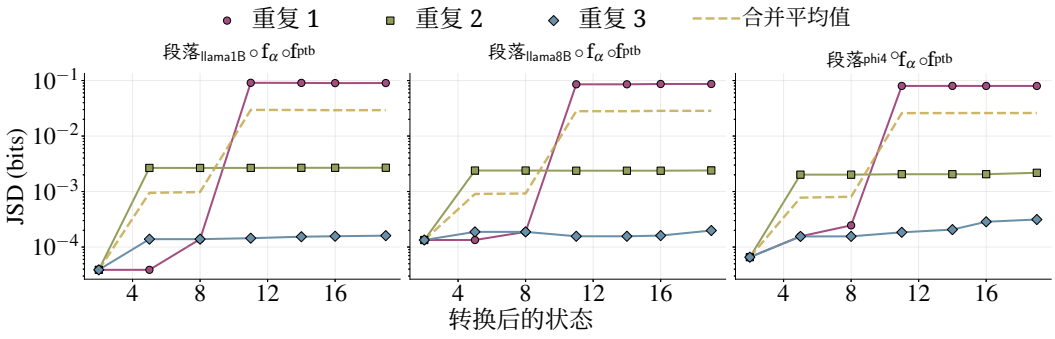


图 19：对于 PTB 转换器，将状态从 IP-通用转换为非IP通用时，每次重复实验的詹森-香农散度（JSD）与已转换状态数量的关系（τ =1e-3）。每次重复实验均采用不同的随机转换顺序。突增是由个别高连通性状态造成的；转换这些状态会导致 JSD 突增。

H 基于状态的分解

正文中介绍的算法会枚举源 字符串：商 $\mathcal{Q}(y)$ 和余项$\mathcal{R}(y)$ 分别表示为显式的字符串集合。当这些集合为无限集时，如果不进行剪枝，字符串枚举方法将无法终止。

另一种方法是处理预覆盖 DFA 的状态空间，而非其字符串空间。由于预覆盖 DFA P_y (§5.1)只有有限个状态，基于状态的遍历总会终止，即使 $\mathcal{Q}(y)$ 或 $\mathcal{R}(y)$ 是无限的。关键思路是将商与余项表示为 DFA Q 和 R——它们分别接受（可能无限的）集合 $\mathcal{Q}(y)$ 和 $\mathcal{R}(y)$——而不是枚举集合中的元素。随后便可通过 $\mathcal{P}(y) = \langle[\![\mathrm{Q}]\!]\rangle \sqcup [\![\mathrm{R}]\!]$ 恢复完整的预覆盖。
图 20 中的算法对 P_y 的状态执行广度优先搜索。在每个状态 s，它都会检查s 是否为全称——也就是说，以 s 为根的子自动机是否接受 $\mathcal{X}^*$ （参见 is_柱集）。如果是， s就会被标记为商成员和接受状态，并且不会继续探索其后继状态，因为全称状态的所有扩展都已覆盖。如果 s 是接受状态但并非全称状态，就会被标记为余项成员和接受状态，并继续探索其出弧。

结果是一对确定有限自动机，它们具有相同的状态空间和截断弧集，但接受状态不同。商自动机 Q 将普遍状态设为接受状态，并删除从普遍状态出发的弧（因为广度优先搜索不会继续扩展这些状态）；因此 $[\![\mathrm{Q}]\!] = \mathcal{Q}(y)$恰好接受商元素。余数自动机 R 使用相同的截断弧集，但仅将非普遍接受状态标记为接受状态；因此， $[\![\mathrm{R}]\!] = \mathcal{R}(y)$。

```
247 def dfa_分解(f y):
248     I
        P_y ← trim(d项    inize(proj_X(
        y                           y)*
                                        268
                                        ))) 249 q ← 队列(I )
270
        ← ∅                             272
250     V ← ∅           ← ∅ ; 弧 Q ; R 定义 is_柱集(s): 269    V ← {s}; q ← 队列({s})
251       ← ∅
当 q: 271       s ← q.pop() 如果  s ∉ F_y: 252 当 q: 253       s ← q.pop() 254 如果
 s ∈ V: continue 273 返回 False 274 对于  x ∈ X: 275         s′ ← 步数_y(s, x)
255       V.add(s) 276 如果  s′ = ∅: 返回 False 256 如果  s ∈ F_y: 257 如果 is_柱集
(s): 277 如果  s′ ∉ V: 278              V.add(s′); q.push(s′) 258 Q.add(s) 279 返回
True 259 continue # 不要扩展 260 else: 261 R.add(s) 262 对于  x ∈ X:
263         s′ ← 步数_y(s, x) 264 如果  s′ = ∅: continue 265 q.push(s′) 266
arcs.add(s --x--> s′) 267 返回 DFA(V,arcs,Q), DFA(V,arcs,R)
```

图20：将预覆盖基于状态分解为确定有限自动机。 左：对预覆盖 DFA 进行广度优先搜索P_y。普遍状态在 Q 中变为接受；非普遍接受状态在 R 中变为接受。离开普遍状态的弧不会被收集。 右： is_柱集与图 3 中的普遍性广度优先搜索相同，但直接接收 DFA 状态，而不是通过run$_y$从源字符串计算该状态。

与基于字符串的算法的比较。 基于状态的算法总是在有限时间内终止——即使当 $\mathcal{Q}(y)$ 或 $\mathcal{R}(y)$ 为无穷时——因为 P_y 只有有限多个状态。所得的确定有限自动机Q 和 R 为这些潜在的无穷集提供了紧凑的有限表示。

然而，实现自回归接口（§ C.1）仍需枚举这些机器所接受的字符串：根据公式（8a）计算 $\overrightarrow{p_{\mathcal{Y}}}(y)$，需要对 $p_{\mathcal{X}}(x)$ 中的元素求和；而当这些集合是无限集时，无论如何都必须截断求和。因此，尽管 DFA 表示保证了有限分解，但其本身并不能实现有限时间评分。

I 相关工作

现代语言模型定义了词元序列上的概率分布（见 §2）。为提高效率并管理词汇表（亦称字母表），它们通常依赖 BPE（Sennrich 等人，2016年；Gage，1994年）或单元语法（Kudo，2018）等子词方案。尽管这些方法取得了显著成功，但其基本单位往往与语言边界并不一致，而且在语言模型下，任意给定的字符串通常都对应着数量呈指数级增长、且概率质量非零的分词变体。近期研究通过强制采用规范分词，消除非规范编码中的概率质量，以应对这一问题（Vieira 等人，2025b）；而Geh 等人(2024) 则表明，汇总非规范分词选择的概率质量能够提供有用信号，从而提升下游准确率。

子词切分还会引发提示边界问题（Vieira 等人，2025a），即对提示末尾字符做出难以察觉的改动（例如追加一个空白）就可能使编码后的词元序列在词元空间中走上完全不同的路径，导致模型放弃原本概率很高的续写。为克服这些问题，Vieira 等人（2025a）提出了一种算法，用于将基于词元的语言模型转换为基于字符的语言模型。尽管其贡献以字符为中心，但其基本思想可以加以推广（如本文所示）。

许多应用需要一种方法，将在子词词元上学习到的概率质量准确转换到其他类型的单元，例如自然语言处理中的字节、单词或语素，或计算生物学中的氨基酸，如§1所述。心理语言学中还有一个例子：研究者常常需要细粒度的惊讶度估计，例如根据单词前三个字符的可预测程度来预测读者跳读该单词的可能性（Rayner 等人，1982；Blanchard 等人，1989）。为此，近期研究已着手应对子词分词带来的挑战（Nair& Resnik，2023；Beinborn & Pinter，2023；Pimentel&Meister，2024；Oh &Schuler，2024；Giulianelli 等人，2024）。例如，Oh & Schuler （2024）和Pimentel & Meister（2024）认为，前导空白分词会在惊讶度估计中引入混杂因素，并转而主张将尾随空白的概率纳入此类计算。

此外，Pimentel 与 Meister（2024）提出了一种专门的方法，用于将基于词元的语言模型转换为基于单词的语言模型。然而，他们的方法未能建模英语分词对上下文的敏感性，例如，它对例（1）中的两个句点一视同仁，而英语正字法并非如此。此外，该方法的合理性论证要求存在一组特定的词尾标记，它们（如果存在）必须出现在词元末尾。现在，我们来探讨如何构造这样一个转换器。设$\mathtt{f}_\alpha$ 是一个将词元字母表转换为字符字母表的转换器， D 是分隔符集合。转换器$\mathtt{f}_D$如图21所示。给定一个定义在 $p_\mathcal{X}$ 之上的语言模型 $\mathcal{X}$，我们便可将它们组合成一个转换器 $p_\mathcal{X} \circ \mathtt{f}_\alpha \circ \mathtt{f}_D$，从而得到一个以分隔符分隔的单词上的转导语言模型。然而，这种做法相当天真。遗憾的是，基于分隔符的分隔无法区分句点何时应作为独立符号，何时不应如此，例如例（3）。

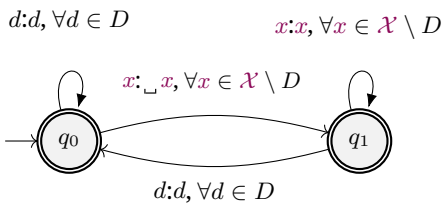


图21：一种简单的有限状态转换器，可在不使用上下文信息的情况下将字符流切分为单词，并在每个单词的开头插入。

基于分隔符的方法对于大多数基于字节对编码的语言模型也会失效，这是因为分隔符候选项聚！！！类在了一起。例如，GPT-4o 的字母表包含词元 $_{10880}$, , 它由

！！！完全由词尾符号组成。比如，按照 PTB 指南，10880 应拆分为三个连续的正字法词!!!。相比之下，我们认为，心理语言学建模所适用的分词方案应根据研究目标来确定，而不应依据任何特定分词器的特性来确定。

分词挑战并非自然语言建模所独有。在计算生物学中，DNA、RNA和蛋白质序列是小字母表上的长而未分段的字符串，给分词带来了挑战。因此，研究人员会在不同的分词方案之间交替使用，例如k-mer、学习得到的子词和基序感知分段器（Ji等人，2021年；Nguyen等人，2023；Dotan等人，2024；Wang等人，2024；Qiao等人，2024）。由于语言模型通常在不同的分词方案下进行训练，其自回归预测——逐步概率和条件分布——在不同分词方案之间无法直接比较。

基于转换器的分词方法已经相当成熟——WordPiece（Wu等人，2016年）可以实现为转换器（Song等人，2021年），并且已为BPE构建了确定性有限自动机（Berglund和van der Merwe，2023；Berglund等人，2024）。此外，转换器在语言建模中也有悠久的历史（Mohri，1997），并已被用于受约束解码，其中有限状态转换器用于强制执行词汇或结构约束（Allauzen等人，2014；Ghazvininejad等人，2016年；Stahlberg等人，2019年；Willard和Louf，2023；Koo等人，2024；Cognetta等人，2025年）。

与之密切相关的是神经有限状态转换器（Lin 等人，2019年）以及更早期的神经有限状态转换器混合模型（Rastogi 等人，2016年），它们为转移赋予（神经）权重，并将前缀/序列概率计算为与给定输出一致的所有路径的路径和。我们的构造也可以用类似的方式理解：我们对所有映射到给定输出的源字符串（路径）进行边缘化。不过，我们并非学习转移权重，而是专注于转换现成的预训练语言模型，从而通过商余分解实现高效推理。

在本研究中，我们将字符级转换加以推广，并将 Vieira 等人(2025a) 扩展为一个框架，使我们能够将一种语言模型转换为另一种语言模型，而不再局限于严格前缀单调映射的有限情形。只要集合之间的映射可以由有限状态转换器描述，我们便支持不同单位集合之间的转换以及保持单位的变换。

J 局限性

实证范围。 尽管从理论上说，只要给定有效的转换器，我们的框架就能将任何语言模型转导为任意目标单位，但我们仅测试了有限几种架构（GPT-2、LLaMA 3 和 Phi-4）以及特定目标单位（字节、宾州树库词元和氨基酸）。未来研究可以将分析扩展到更广泛的模型、数据集和单位。

表达能力。 我们的分析主要集中于函数式有限状态转换器和正则语言。未来的工作可以考虑动态构建的转换器，以及分布于表达能力更强的语言上的分布；还可以考虑由非函数型转换器编码的随机映射——在这种情况下，需要调整全称性的概念。

近似质量。 当源语言模型将其大部分概率质量集中在少量前缀上，而这些前缀在转换器下映射到当前目标前缀时，我们基于剪枝的推理算法表现良好——也就是说，剪枝后商与余数的有效规模较小。当概率质量分散在许多源前缀上时（例如 DNA 到氨基酸的情况），剪枝必须舍弃总概率中更大的比例，近似结果会变差。重要性采样方法（§C.3）可以提供一种系统性偏差较小的替代方案。

速度。 我们的算法和实现的速度可能无法满足所有使用场景。目前，要获得完整分布（例如用于解码或模型比较），速度约需达到 10–20 字节/秒。虽然这对许多任务而言并非不可接受，但仍有很大的改进空间。

边缘化。 转导语言模型的一项核心特性是边缘化：转导会对源字符串的概率求和，以计算目标字符串的概率，并汇总所有映射到同一目标的源字符串的概率质量。由于自然语言允许用多种方式表达相同的含义，转导可以对这些表达进行规范化，从而在感兴趣的空间中得到更具代表性的输出分布。以一道数学题为例：“如果有 12 名学生比汤姆年纪小，另有 12 名学生比他年纪大，那么总共有多少名学生？”答案可以是“25”，也可以是“二十五”，或许还可以是“12+1+12”。如果这些答案中任何一个具有显著的概率质量，我们就希望将它们相加。在思维链推理中，我们也可以设想对系统生成的多个样本的概率进行求和并归一化，这种方法称为自洽性(Wang 等人，2023）。我们期待未来的研究能将转导语言模型应用于这类场景。