TurboBias：由 GPU 加速的短语提升树驱动的通用 ASR 上下文偏置

Andrei Andrusenko*‡, Vladimir Bataev*, Lilit Grigoryan*, Vitaly Lavrukhin† and Boris Ginsburg†
*NVIDIA, Yerevan, Armenia
†NVIDIA, Santa Clara, USA
‡Corresponding author, aandrusenko@nvidia.com

摘要—识别特定的关键短语是上下文化自动语音识别（ASR）中的一个重要任务。然而，大多数现有的上下文偏置方法都存在需要额外模型训练的局限性，显著减慢了解码过程速度，或者限制了 ASR 系统类型的选取。本文提出了一种支持所有主要类型（CTC、转导器和注意力编码-解码模型）的通用 ASR 上下文偏置框架。该框架基于 GPU 加速的词提升树，使其能够在贪婪搜索和束搜索解码中以浅层融合模式使用，并且即使在大量关键短语（多达 20K 项）的情况下也不会有明显的速度下降。实验结果表明所提出方法具有很高的效率，在准确性和解码速度方面超过了考虑的开源上下文偏置方法。我们的上下文偏置框架作为 NeMo 工具包的一部分进行了开源。

***Index Terms*—自动语音识别，上下文偏置，短语提升，贪婪解码**

I. 介绍

现代端到端的自动语音识别（ASR）系统，如连接时序分类（CTC）[1] 、递归神经转导器（RNN-T）[2] 和注意力编码-解码器（AED）[3] 已经在常见数据领域实现了相对较高的语音识别准确率 [4] 。然而，这类模型在识别训练数据集中罕见或不存在的特定单词/短语（如联系人姓名、产品标题、技术术语等）时常常存在问题。使用上下文偏置方法来解决这个问题。

上下文偏置方法的关键点是使用特定于目标领域的附加数据。这可以是一组由句子组成的文本语料库，或一个需要提高识别准确性的关键词/短语列表。本文考虑了基于短语增强的上下文偏置方法的第二种变体，因为在实际语音识别使用场景中准备可能出现在系统操作中的关键短语的上下文列表相对容易。然而，收集完整的文本数据可能会很困难。

上下文偏置方法可以提高关键词识别的准确性，但这一过程带来了额外的限制。例如，深度融合方法意味着将上下文信息引入 ASR 模型中。这个过程需要重新训练 ASR 模型（交叉注意力方法）[5], [6] 或者训练一个额外的上下文模块 [7]–[9]。SpeechLM 模型也支持作为额外提示引入上下文信息 [10], [11]，但这同样需要在模型训练过程中使用情景学习。

浅融合方法允许避免额外的模型训练 [12], [13] 。在这种情况下，在解码阶段应用了上下文偏置，增加了从集成到辅助增强树或图中的上下文列表中识别关键短语的概率。在 [14] 中显示，浅融合在关键词识别准确性上仅略逊于深度融合，同时保持了使用的灵活性。

浅层融合方法的缺点是解码过程速度显著降低，必须在束搜索模式下进行以扩展假设搜索空间。这个问题在解码 RNN-T 和 AED 模型时尤为严重，因为在束搜索过程中对解码器模块的调用次数与贪婪模式相比显著增加。对于 RNN-T 而言，解决这一问题尤其重要，因为该模型常被选为准确率性能、内部语言模型（LM）能力和流式支持之间的权衡 [15], [16]。

为了加快上下文偏置过程，可以使用基于 CTC 的词检测器 [17]，结合来自 CTC 或 RNN-T 模型的贪婪解码结果与检测到的关键字。然而，这种方法需要一个 CTC 模型，并且仅适用于离线解码。另一种加速 RNN-T 模型浅层融合的方法可以是使用 GPU/TPU 友好型实现的 Knuth-Morris-Pratt（KMP）模式匹配算法 [18]。所提出方法的缺点是其对其他 ASR 模型的应用有限，且缺乏开源实现。

最近使用标签循环算法 [19] 结合 CUDA 图形实现 [20] 加速 RNN-T 解码的研究已经将贪婪解码和束搜索解码之间的差距缩小到了最小，接近了贪婪 CTC 解码的速度。这使得在 GPU 上完全实现统计 n 元语言模

型（NGPU-LM）[21] 的浅层融合成为可能，在贪婪和束搜索 [22] 解码模式下均不会显著减慢速度。该方法有利于特定数据领域的标准上下文偏向任务，但 n 元 LM 在提升特定孤立词时表现不佳。此外，n 元 LM 需要来自目标域的训练文本语料库。

本工作介绍了 GPU-PB，这是一个支持所有主要 ASR 模型类型的通用上下文偏置框架：CTC、转导器（RNN-T）和 AED。该框架基于一个具有修改后评分权重分布的 GPU 加速短语增强树。这一关键技术革新使得在束搜索和贪婪解码模式下都能实现高精度性能，同时仅带来极小的计算开销（2-5% RTFx）。获得的结果显示了所提出方法的优越效率，即使包含大量关键词组（高达 20K），其在语音识别方面的准确性和解码速度也超过了考虑中的开源上下文偏置方法。

我们的主要贡献如下：

- **一个新的上下文偏置框架** 用于快速短语提升，在并行计算环境中效率高。
- **修改后的权重分布于提升树中** 用于贪心和束搜索中的高精度浅层融合。
- **支持所有类型的 ASR 模型**：CTC, 转导器（RNN-T）和 AED。

我们的上下文偏置框架在 NeMo 工具包 [1],[2] 中公开可用。

II. 方法

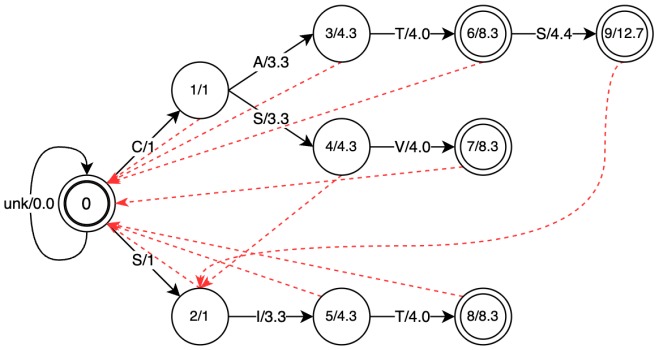
A. 浅层融合背景

标准的上下文偏置过程在浅层融合模式下可以表示如下：

$$W = \arg\max_{W} \log P(W|X) + \lambda \log P_C(W) \qquad (1)$$

这里，W 是基于输入声学特征 X 的 ASR 模型识别结果（单词或标记）。P_C 表示应用了权重为 λ 的上下文偏向模型。P_C 模型可以是一个统计 n 元语言模型、神经网络语言模型或短语增强树/图。

Aho – Corasick 算法 [23] 可以用于构建适用于此上下文偏置任务的短语增强树。这样的树是基于从上下文列表中提取的短语进行分词后所构造的前缀树。每个树节点都是带有相应得分的标记。失败链接（或在 n 元语法语言模型中的回退）允许在当前树节点中缺少所需转换标记时减去累计得分。



T	H	E	C	A	T	I	S	S	I	T	T	I	N	G
0	0	0	1	3.3	4	0	1	-1+1	3.3	4	-8.3	0	0	0

图 1. 一个用于单词 “CAT, CATS, CSV, SIT” 的字符级分词的提升树示例。树下面是当解码 “the cat is sitting” 序列时如何分配提升分数的一个示例。

一个基于 Aho – Corasick 算法的短语提升树的开源 Python 实现示例在 Icefall 工具包 [3] 中进行了展示。该实现在 RNN-T 束搜索解码中用于浅层融合。然而，这种实现限制了请求的数量，每个查询只允许一个令牌和一种状态，对解码速度产生了显著影响。为了高效运行，有必要每次请求都获取整个词汇表的概率分布（例如，对于使用 BPE 标记化的我们的模型来说是 1024 个 [24]）。此外，基本实现仅限于在 CPU 上使用。

之前提出的 NGPU-LM [21] 允许您在整个词汇表上执行查询，这是由于其存储和遍历树的高效结构。此外，这项工作具有一个自定义的 Triton 内核，使它能够与 CUDA 图在浅融合模式下运行贪婪和束搜索解码。然而，这种结构设计用于利用 n-gram 语言模型，这不同于短语增强任务。即使我们只为上下文列表中的短语构建了一个语言模型，树中也会包含许多多余的转换（例如，所有可能的单字词作为短语的有效起始）及其回退。这种结构性限制最终导致了关键短语识别准确率下降，特别是在偏置列表广泛的情况下。

B. 提议的方法

为了解决 NGPU-LM 中通过浅层融合和无效转换引起的短语提升速度问题，我们提出了一种新的 GPU-

[1]https://github.com/NVIDIA/NeMo/pull/14108

[2]https://github.com/NVIDIA/NeMo/pull/14277

[3]https://github.com/k2-fsa/icefall/blob/master/icefall/context_graph.py

PB 方法，该方法结合了 NGPU-LM 的高效 GPU 树架构与短语提升树，并采用了修改后的权重分布。

在 GPU-PB 编译的第一步中，我们基于 Aho – Corasick 算法构建一个标准前缀树，用于上下文列表中的标记化短语。然而，在树中的权重分布均匀且仅在最终节点（如 Icefall 实现）附加奖励（所有转换的累积分数）的情况下，贪心搜索可能会导致低效率。在这种情况下，在到达树中最终短语节点之前没有支持多个备选假设的可能性。每个时刻，贪心搜索只选择一个假设。评分值应基于树的深度进行增加，以确保当每次解码步骤只有一个假设时能够遍历提升树。

为了更高效地分布得分，我们将所有深度大于一的树转换得分通过对数依赖关系 $c_0 * \beta + \ln(depth)$ 显著增加。这里，c_0 是默认上下文得分（例如，1），β 是深度缩放参数（例如，2）。这种方法使我们能够近似 n-gram 语言模型中的权重分布（对于正得分），并将其应用于贪婪搜索和束搜索解码。

接下来，我们将获得的前缀树转换为基于 NGPU-LM 的有效树结构。对于根节点，添加了一个带有未知标记分数的自环（默认为 $c_{unk} = 0$），针对所有不是来自上下文列表短语有效开头的标记（它们没有从初始前缀树根部的过渡）。将其他所有前缀树弧线转换为一个新的基于 GPU 的树，支持按元组 “(from_state, token)” 排序。每个节点还具有 start_arcs 和 end_arcs，最终允许使用 “index-select” 操作以实现高效的分数检索。

我们将每个节点的所有回退路径保存下来，计算此转换的得分作为当前节点累积得分与回退指向节点得分之间的差值。这允许减去从不正确的部分提升中获得的部分分数。我们还为树中的每个最终节点（短语结尾）设置零分的回退以防止额外的分数扣除。上下文词的前缀树示例如图 1 所示。

获得的基于 GPU 的结构使得在推理过程中可以通过索引选择操作在整个词汇表中使用查询。额外的自定义 Triton 内核进一步加速了树的操作，使其可以与 CUDA Graph 一起用于 CTC 和 RNN-T 解码。此外，GPU 结构允许以最小的解码速度开销使用许多上下文短语。这也可以用于添加不同的词形（例如复数形式）和大小写（现代模型默认支持它）。所有这些显著增加了树中的加速短语数量。整个 GPU-PB 管道如算法 1 所示。

C. 适应性调整对于自动语音识别模型

GPU-PB 具有与 NGPU-LM 类似的结构。提出的短语增强方法也可以应用于贪婪和束搜索解码模式下的所有主要 ASR 模型，包括 CTC、Transducers 和 AED，使用高效的解码管道与 NGPU-LM [21], [22], [25]。

对于 CTC 和 RNN-T 模型的贪婪解码，每个解码步骤都会执行两阶段标记选择 [21]。首先，在不进行加权的情况下应用贪婪选择。如果结果为空白标记，则继续下一个解码步骤。对于 CTC 模型，我们还会保留所选的重复标记，而不对其应用加权权重，并进入下一步。否则，我们将对剩余的标记（RNN-T 模型为非空白标记，CTC 模型为非空白且不重复的标记）使用提升树进行重新加权，并从此类别中贪婪地重新选择输出。

在 AED 模型中，句尾（EOS）符号用作辅助标记，控制解码过程的结束。NGPU-LM 可以直接预测解码过程中的最终权重，因为它是一个完整的统计语言模型。对于 GPU-PB，在我们没有方法从提升树中获得公平的 EOS 分数的情况下。在这种情况下，来自提升树的得分可能会超过 EOS 得分，这可能导致 AED 在句子结尾处出现解码幻觉。为了解决这个问题，我们使用以下启发式方法——在每个解码步骤中，我们将默认的 EOS 得分增加通过提升树获得的最大得分。当我们从提升树中检测到整个短语时，我们也增加节点最终权重值的 EOS 得分。这些技术使我们能够应对 AED 模型中的 EOS 抑制问题。

另一种使用 GPU-PB 的贪婪解码特征是可能降低未包含在上下文列表中的单词识别准确性。这很可能与由于未知标记和增强树中第一个转换之间的分数差异导致这些单词从增强树中失真有关，受制于几个并行识别假设的缺失。在这种情况下，我们建议使用一个接近树 c_0 中第一个转换分数的未知分数值。

III. 实验设置

A. 自动语音识别模型

为了评估该方法，我们使用了在大量数据上训练的 ASR 模型以接近真实世界场景。为此，我们利用了一个包含 24,000 小时带标签的英语语音的开放数据集以及一个具有 1024 个标记的 BPE 分词器 [24]。

作为 ASR 模型，我们使用相同的 FastConformer 编码器架构 [26] 训练了 CTC、RNN-T 和 AED（Ca-

nary [27]）模型，参数量为 108M。为了方便训练和通用使用 CTC 和 RNN-T 模型，我们训练了一个混合型 Transducer-CTC 模型 [28]（一个与 CTC 和 Transducer 输出头一起训练的共享编码器）。对于 RNN-T 头部，使用了大小为 640 的单层 LSTM 预测网络（解码器），这将总模型参数量增加到 114M。对于 AED 模型，我们利用了一个 4 层 Transformer（FF 1024，Self-attention 256）作为解码器，最终模型参数量为 179M。

模型评估在贪婪和束搜索解码中进行，束大小为 8，除了 AED 模型。对于 AED，我们使用了束大小为 3，因为进一步增加束会导致显著的解码速度减慢。

B. 评估数据

我们使用了多个特定领域的测试数据集来评估上下文偏置任务。

收益 21。Earning21 ~ [29] 表示金融数据领域典型的约 40 小时的收益电话会议。该数据集包含长时间记录，最长可达 1.5 小时，这可能导致许多自动语音识别模型出现识别问题。我们将数据集分割成最长 30 秒的样本片段。作为开发集和测试集，我们分别选择了时长为 5 小时和 10 小时的子集。对于上下文列表，我们使用了基准预言列表 [4]，并移除了长度为 2 个字母的短语以防止此类情况下的误接受率增加。结果，我们获得了包含 893 个短语的上下文列表。

多媒体. Additionally, we prepared a subset from the MultiMed medical dataset [30]. We further processed the test and dev sets by filtering out files with incorrect transcriptions based on the recognition results of the base model (WER > 30%). The final dataset sizes were 13.5 hours for the test and 7.3 hours for the dev. To select specific boosting phrases, we used the LLaMA 3.3-70B Instruct model [31], prompted to tag medical terms, such as diseases, procedures, medications, and clinical conditions. To identify named entities such as personal names, brand names, and organizations, we used the spaCy toolkit [32]. We filtered the resulting phrases based on the recognition results of the baseline ASR model, eliminating simple (>70% accuracy) and low-frequency (<10 occurrences) phrases. We obtained a list of 991 unique phrases.

CSTalks. 我们收集了来自科技公司的开放主题演讲，其中包含许多特定术语，以测试在现代计算机科学领域中提出的方法。在文本处理和音频分段之后，我们获得了用于开发集的 3.2 小时和用于测试集的 8.3 小时。我们使用了与 MultiMed 数据集相同的方法来准备短语增强列表。生成的上下文偏差列表包含 200 个独特的单词，这些单词在这些集合中具有很高的出现密度。

表 I 提供了用于评估数据的一般信息，并进行了详细说明。开发集用于提升参数的搜索，测试集则用于获取第 IV 节中的最终结果。

C. 指标

我们使用 F 分数 ($2 * Precision * Recall/(Precision + Recall) * 100\%$) 来衡量关键短语识别的准确性，该分数是根据上下文列表中的短语计算得出的。此外，我们还计算了整个识别结果的整体 WER 值，以评估 ASR 系统的整体识别质量。为了追踪解码速度，我们测量了逆实时因子 (RTFx)。RTFx 是通过将总音频时长除以 ASR 模型评估时间来计算的。RTFx 在一次热身运行之后进行测量，是在单个 NVIDIA RTX A6000 GPU 上使用 32 的批量大小进行了 3 次运行后的平均值。

D. 与其他方法的比较

为了比较所提出的方法，我们调查了其他支持短语增强的开源解决方案：Pyctcdecode[5]，CTC-WS [17] 和 NGPU-LM [21]。Pyctcdecode 是一个具有浅层融合模式下词增强功能的 CTC 解码框架。CTC-WS 是一种将 CTC/RNN-T 贪婪解码结果与单词探测器检测到的关键短语相结合的方法。单词探测器使用 CTC-logits 在上下文偏置图上以快速束搜索解码模式中查找单词。需要 CTC-logits 的存在要求具有混合的 Transducer-CTC 架构才能与 RNN-T 一起工作。对于 NGPU-LM 方法，我们基于上下文列表构建了一个六元词级别语言模型。

我们还研究了具有均匀权重分布和最终树节点额外假设奖励的 GPU-PB（GPU-PB_{uw}），类似于 Icefall 框架中提升树的基础实现。

[4]https://github.com/revdotcom/speech-datasets/blob/main/earnings21/bias_lists/oracle_list.txt

[5]https://github.com/kensho-technologies/pyctcdecode

IV. 结果

对于在测试集上呈现的所有结果，我们通过实验为每种方法选择了提升权重参数，以在开发集上获得尽可能小的 WER 值。

A. 提议的短语提升方法

提出的上下文偏置方法的整体结果见表 II。GPU-PB 提高了关键短语识别的准确性（F 值），并且在几乎所有考虑的情况下都实现了整体语音识别准确性的额外提高（WER）。这种效果在 CSTalks 数据集中最为明显，因为该数据集具有来自上下文列表的关键短语最高密度，因此可以改善其识别准确性。

贪心算法与束搜索. GPU-PB 的一个重要特性是在贪婪解码模式下能够提升结果（平均绝对 F 分数提高 8-10%），因为这允许在流式解码中增强短语。为了从 GPU-PB 获得最佳性能，需要使用束搜索（平均绝对 F 分数提高 17-23%），因为在同时处理多个假设的情况下可以扩展从上下文列表中提取关键短语的搜索空间。然而，当前版本的束搜索仅适用于离线解码场景，并且比贪婪解码慢 11-20%。

实时格式化 RTF. GPU-PB 在所有考虑的 ASR 模型和数据集（包括上下文列表大小高达 1000 个短语的测试集 – MultiMed）中的实时因子 (RTFx) 平均减慢了 2-5%，即使在束搜索的情况下也是如此。这一事实证明了使用 GPU 实现短语提升对贪婪解码和束解码的高度效率。

自动语音识别模型. 在准确性与解码速度方面，RNN-T 模型由于存在解码块并且使用 CUDA 图形有效地实现了束搜索，因此是上下文偏向任务中最均衡的选择。GPU-PB 也在贪婪和束搜索解码的 AED 模型中表现出高效率。然而，这种 AED 模型的速度显著低于 CTC 和 RNN-T。尽管如此，这些结果为利用 SpeechLM 模型（该模型采用自回归 Transformer 解码器）与 GPU-PB 提供了前景。

B. 与其他方法的比较

与其他现有的开源解决方案的比较结果见表 III。我们在 CSTalks 数据集上使用 CTC 和 RNN-T 模型评估了所有方法。提出的 GPU-PB 方法在束搜索情况下表现出最高的效率，绝对 F 分数比所考虑的方法高出 2-18%，并且解码速度也很快。

CTC-WS 方法在贪婪解码中表现出最高的准确性，这是因为存在一个额外的单词检测器，需要访问完整的音频文件。这一限制使得 CTC-WS 在流模式下的使用受限。此外，由于单词检测器运行在 CPU 上，CTC-WS 的速度比 GPU-PB 慢大约两倍。

均匀的权重分布对于 GPU-PB（GPU-PB_{uw}）产生了良好的结果。然而，由于假设权重累积过程较长，这种方法的关键短语识别准确性受到限制，在贪婪模式下这一点尤为重要。

Pyctcdecode 相对于基线也显示出改进，其准确率几乎与 GPU-PB_{uw} 的 CTC 准确率相同。然而，随着增强短语数量的增加，pyctcdecode 的速度显著下降，这是由于次优浅层融合实现所致。

使用 NGPU-LM 也提高了准确性，但 n 元语言模型中存在的大量无效转换限制了这种方法的有效性。这一事实在束解码中尤为明显，在束解码中，该方法的结果甚至比贪婪模式下更差。

C. 稳健性

为了评估所提出的 GPU-PB 方法的鲁棒性，我们通过将上下文偏置列表增加到 20K 短语来考察其性能。在这个实验中，我们使用了 CSTalks 数据集以及 RNN-T 模型的束搜索解码模式。我们将束大小保持为 8，并稍微降低了提升权重从 1 到 0.7 以避免在解码过程中出现大量误接受。CSTalks 的基础上下文列表大小是 200 个短语。接下来，我们使用 LLM 模型生成了大约 20K 特殊术语，每个术语长度为 1 至 3 个单词，并逐渐扩展基础上下文列表以包含多达 20K 个短语。

所描述的实验结果如图 2 所示。根据获得的结果，我们可以得出结论：提出的算法对上下文列表大小的变化相对鲁棒。将提升短语的数量从 0 增加到 20,000 仅使整个系统运行时间减慢了大约 5%（蓝色曲线）。识别精度由于额外的误接受而从 81.1%下降到了 78.4% F-score（梯度圆圈），并且字错率（WER）从 9.86%增加到 11.42%（紫色曲线），但仍明显优于基线（44.2% F-score 和 12.83% WER）。

D. 互补性

我们也探讨了在 CTC 束搜索解码中将 GPU-PB 与 NGPU-LM 一起用于 MultiMed 数据的可能性。在这种情况下，使用上下文列表中的短语进行了短语增强。对于语言模型融合，我们基于训练文本数据构建了一个 n

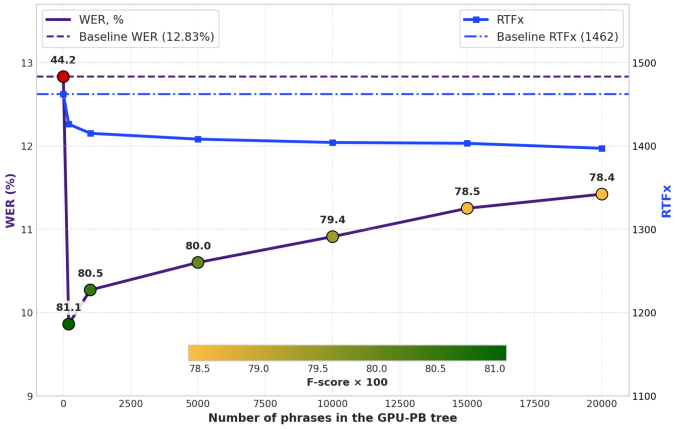


图 2. GPU-PB 对上下文列表大小的鲁棒性。左侧 y 轴表示 WER（紫色）。右侧 y 轴表示 RTFx（蓝色）。虚线代表没有短语提升情况下的基线 WER 和 RTFx 值。F 分数由具有确切值的梯度圆圈在 WER 图表上表示。红色表示最低 F 分数，绿色表示最高。

元语法语言模型。这种组合帮助将测试集上的 WER 从 15.09%降低到 13.55%，而单独使用这些方法分别显示了 14.47%和 14.00%的 WER。结果证明了所考虑方法之间的互补性，从而提高了整体识别效果。

V. 结论

在本文中，我们介绍了一个通用的 ASR 上下文偏置框架，支持所有主要的 ASR 模型，包括 CTC、转导器（RNN-T）和 AED 模型。该框架利用一个 GPU 加速的短语提升树，并采用修改后的评分权重分布以实现高效的浅层融合解码。获得的结果显示关键短语识别（F 分数）在贪婪搜索中提高了 8-10%，在束搜索解码中绝对值提升了 17-23%，并且与没有短语提升的基线相比，整体 WER 也有所改善。使用短语提升的速度开销仅占总 ASR 运行时间的 2-5%RTFx，表明其对上下文列表大小增长（高达 20K 项）具有很高的鲁棒性。所提出的方法在准确性和解码速度上超越了考虑中的开源上下文偏置方法。提出的上下文偏置框架作为 NeMo 工具包的一部分进行了开源。

参考文献

[1] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *ICML*, 2006.

[2] A. Graves, "Sequence transduction with recurrent neural networks," in *ICML*, 2012.

[3] W. Chan, N. Jaitly, Q. V. Le, and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *ICASSP*, 2016.

[4] R. Prabhavalkar, T. Hori, T. N. Sainath, R. Schluter, and S. Watanabe, "End-to-end speech recognition: A survey," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 32, pp. 325–351, 2023.

[5] G. Pundak, T. N. Sainath, R. Prabhavalkar, A. Kannan, and D. Zhao, "Deep context: End-to-end contextual speech recognition," in *SLT*, 2018.

[6] M. Jain, G. Keren, J. Mahadeokar, and Y. Saraf, "Contextual rnn-t for open domain asr," in *Interspeech*, 2020.

[7] X. Yang, W. Kang, Z. Yao *et al.*, "Promptasr for contextualized asr with controllable style," *ICASSP*, 2024.

[8] D. Le, M. Jain, G. Keren, S. Kim *et al.*, "Contextualized streaming end-to-end speech recognition with trie-based deep biasing and shallow fusion," *Interspeech*, 2021.

[9] P. Harding, S. Tong, and S. Wiesler, "Selective biasing with trie-based contextual adapters for personalised speech recognition using neural transducers," *INTERSPEECH 2023*, 2023.

[10] M. Wang, W. Han, I. Shafran *et al.*, "Slm: Bridge the thin gap between speech and text foundation models," *ASRU*, 2023.

[11] Z. Chen, H. Huang, A. Y. Andrusenko, O. Hrinchuk, K. C. Puvvada, J. Li, S. Ghosh, J. Balam, and B. Ginsburg, "Salm: Speech-augmented language model with in-context learning for speech recognition and translation," *ICASSP*, 2024.

[12] D. Zhao, T. N. Sainath, D. Rybach, P. Rondon, D. Bhatia, B. Li, and R. Pang, "Shallow-fusion end-to-end contextual biasing," in *Interspeech*, 2019.

[13] N. Jung, G. min Kim, and J. S. Chung, "Spell my name: Keyword boosted speech recognition," in *ICASSP*, 2022.

[14] R. Huang, M. A. Yarmohammadi, S. Khudanpur, and D. Povey, "Improving neural biasing for contextual speech recognition by early context injection and text perturbation," *Interspeech*, 2024.

[15] Y. He, T. N. Sainath, R. Prabhavalkar, I. McGraw, R. Álvarez *et al.*, "Streaming end-to-end speech recognition for mobile devices," *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6381–6385, 2018.

[16] J. Li, Y. Wu, Y. Gaur, C. Wang, R. Zhao, and S. Liu, "On the comparison of popular end-to-end models for large scale speech recognition," in *Interspeech*, 2020.

[17] A. Andrusenko, A. Laptev, V. Bataev, V. Lavrukhin, and B. Ginsburg, "Fast context-biasing for ctc and transducer asr models with ctc-based word spotter," *Interspeech*, 2024.

[18] W. Wang, Z. Wu, D. Caseiro, T. Munkhdalai *et al.*, "Contextual biasing with the knuth-morris-pratt matching algorithm," *Interspeech*, 2024.

[19] V. Bataev, H. Xu, D. Galvez, V. Lavrukhin, and B. Ginsburg, "Label-looping: Highly efficient decoding for transducers," in *2024 IEEE Spoken Language Technology Workshop (SLT)*, 2024, pp. 7–13.

[20] D. Galvez, V. Bataev, H. Xu, and T. Kaldewey, "Speed of light exact greedy decoding for rnn-t speech recognition models on gpu," in *Interspeech 2024*, 2024, pp. 277–281.

[21] V. Bataev, A. Andrusenko, L. Grigoryan, A. Laptev, V. Lavrukhin, and B. Ginsburg, "NGPU-LM: GPU-Accelerated N-Gram Language Model for context-biasing in greedy ASR decoding," *Interspeech*, 2025.

[22] L. Grigoryan, V. Bataev, A. Andrusenko, H. Xu, V. Lavrukhin, and B. Ginsburg, "Pushing the limits of beam search decoding for transducer-based ASR models," *Interspeech*, 2025.

[23] A. V. Aho and M. J. Corasick, "Efficient string matching: an aid to bibliographic search," *Commun. ACM*, vol. 18, 1975.

[24] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 2016.

[25] L. Grigoryan, V. Bataev, N. Karpov, A. Andrusenko, V. Lavrukhin, and B. Ginsburg, "Flexctc: Gpu-powered ctc beam decoding with advanced contextual abilities," *accepted to ASRU*, 2025.

[26] D. Rekesh, N. R. Koluguri, S. Kriman, *et al.*, "Fast Conformer with linearly scalable attention for efficient speech recognition," in *Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2023.

[27] K. C. Puvvada, P. Żelasko, H. Huang *et al.*, "Less is more: Accurate speech recognition & translation without web-scale data," in *Interspeech*, 2024.

[28] V. Noroozi, S. Majumdar, A. Kumar, J. Balam, and B. Ginsburg, "Stateful conformer with cache-based inference for streaming automatic speech recognition," *ArXiv*, 2024.

[29] M. D. Rio, N. Delworth, R. Westerman, M. Huang, N. Bhandari, J. Palakapilly, Q. McNamara, J. Dong, P. Zelasko, and M. Jette, "Earnings-21: A practical benchmark for asr in the wild," 2021.

[30] K. Le-Duc, P. Phan, T.-H. Pham, B. P. Tat, M.-H. Ngo, C. Ngo, T. Nguyen-Tang, and T.-S. Hy, "Multimed: Multilingual medical speech recognition via attention encoder decoder," *arXiv preprint arXiv:2409.14074*, 2024.

[31] M. AI, "LLaMA 3.3 70B Instruct," 2024. [Online]. Available: https://hf.co/meta-llama/Llama-3.3-70B-Instruct

[32] M. Honnibal, I. Montani, S. Van Landeghem, and A. Boyd, "spacy: Industrial-strength natural language processing in python," *Zenodo*, 2020.

Algorithm 1: 三阶段算法通过提出的 GPU-PB 方法对短语进行增强

Input : Context list $\mathcal{L}$ with tokenised phrases; context score c_0, depth scaling β; unknown-token score c_{unk} (0 by default); shallow fusion weight λ.

阶段 1 `// build a prefix tree` PT `based on the Aho-Corasick algorithm with logarithmic depth bonus`

根节点的初始化 n_{root};

for *phrase* $\pi \in \mathcal{L}$ **do**

  $n_{cur} \leftarrow n_{root}$ `// define current node;`

  for *token,depth in* ***enumerate****(* π *)* **do**

    if $t \notin n_{cur}.next\_arcs$ **then**

      create a n_{new} with arc transition;

      $arc.score \leftarrow \begin{cases} c_0, & depth = 1 \\ c_0 * \beta + \ln(d), & depth > 1 \end{cases}$;

      `// define accumulated node score;`

      $n_{new}.acc\_score \leftarrow n_{cur}.acc + arc.score$;

      $n_{cur} \leftarrow n_{new}$;

    else

      $n_{cur} \leftarrow n_{cur}.next\_arcs[t]$;

  mark n_{cur} as *final node*;

for *node* n *in* PT *with BFS order* **do**

  `// Add failure (backoff) links via BFS;`

  set fail(n) as longest proper suffix in PT ;

阶段 2 `// convert` PT `to efficient GPU structure`

初始化表格弧段;

添加根自环 $(0, \text{UNK}, c_{\text{UNK}})$;

for *arc in* PT *with first order* **do**

  append *arc* parameters to `ARCS`;

  $backoff_{score} = -n_{cur}.acc\_score$;

为词汇表中未出现在 $PT.root.arcs$ 中的所有标记添加具有 c_{unk} 分数的自环转换 ;

for *arc in* PT *with next order* **do**

  append *arc* parameters to `ARCS`;

  $backoff_{score} = n_{cur}.fail.acc\_score - n_{cur}.acc\_score$;

按 *(初始状态, 令牌)* 排序弧段并上传到 GPU;

$BT \leftarrow$ PyTorchnn 模块图结构包装器

阶段 3 `// ASR model inference`

初始化 BT 状态 S;

for *decoding step* **do**

  $\boldsymbol{t}_{1:B} \leftarrow$ last tokens of current beam (or best) hypotheses $hyps$;

  if *GPU-PB enabled* **then**

    $BT_{scores}, S_{next} \leftarrow BT.get\_scores(S[\boldsymbol{t}_{1:B}])$;

    $hyps\_expansion\_score \leftarrow acoustic_{scores} + \lambda \cdot BT_{scores}$;

  else

    $hyps\_expansion\_score \leftarrow acoustic_{scores}$;

  `// Select one or beam best hypotheses;`

  $hyps = prune\_expanded(hyps, hyps\_expansion\_score)$;

表 I
评估数据集的统计。

度量	CSTalks	收益 21	多媒体
Dev set (h)	3.2	5.2	7.3
Test set (h)	8.3	10.0	13.5
Context list size	200	893	991
Occurrences in test	3500	560	4541

示例短语	`gpu, cpu`
	`PyTorch`
	代码助手
		诺基亚
		德勤
		格伦洛克
			主动脉
			针灸
			淋巴管

表 II

所提出的 GPU-PB 方法在 CTC、RNN-T 和 AED 模型的上下文偏置任务中贪婪搜索和束搜索解码模式下的性能评估。GPU-PB 列负责启用短语增强功能。精度和召回率已四舍五入以方便感知。F-SCORE 和 WER 以%为单位；RTFx 是所有集合平均的逆实时因子。

模型	解码	图形处理单元	CSTalks		收益 21（10 小时）		多媒介		实时浮点数↑
		PB	F 分数（精确率/召回率）↑	WER↓	F 值（精确率/召回率）↑	WER↓	F 值（精确率/召回率）↑	WER↓	平均值
CTC	贪婪	–	35.0 (97/21)	13.7	45.7 (94/30)	15.6	54.0 (95/38)	15.0	2181
		✓	64.8 (94/50)	11.9	53.5 (92/38)	15.6	60.2 (93/45)	14.9	2067
	束	–	35.0 (97/21)	13.8	45.7 (94/30)	15.6	54.0 (95/38)	15.0	1874
		✓	**83.2** (90/77)	**10.2**	**67.8** (89/55)	**15.5**	**71.8** (89/60)	**14.3**	1786
递归神经网络转录器	贪婪	–	42.5 (96/27)	12.8	56.0 (95/40)	15.1	60.4 (95/44)	13.9	1822
		✓	70.4 (92/57)	10.7	63.3 (93/48)	15.0	66.3 (91/52)	13.7	1751
	光束	–	44.2 (97/29)	12.8	55.8 (93/40)	14.3	62.5 (95/47)	13.6	1466
		✓	**82.9** (90/76)	**9.6**	**74.0** (88/64)	**14.2**	**75.8** (89/66)	**12.9**	1420
AED	贪婪	–	52.6 (97/36)	12.7	54.7 (92/39)	15.4	64.0 (94/49)	14.1	356
		✓	75.6 (93/64)	10.4	63.8 (91/49)	15.3	69.3 (89/57)	13.9	350
	束	–	53.7 (97/37)	12.5	54.6 (92/39)	15.2	65.7 (94/51)	13.7	145
		✓	**82.4** (94/73)	**10.2**	**66.2** (88/53)	**15.1**	**75.5** (88/66)	**13.1**	141

表 III

所考虑的上下文偏置策略在短语增强任务中的性能比较。F 分数显示精确度/召回率；WER 以百分比表示；RTFx 是逆实时因子。

解码	C-偏置	F 值（精确率/召回率）↑	WER↓	实时文本↑
CTC				
贪婪	–	35.0 (97/21)	13.7	2232
	CTC-WS	**79.8** (90/72)	**10.9**	906
	NGPU-LM	58.5 (96/42)	11.9	2123
	GPU-PB$_{uw}$	53.7 (96/37)	13.2	2181
	GPU-PB	64.8 (94/50)	11.9	1991
光束	–	35.0 (97/21)	13.8	1883
	Pyctcdecode	74.0 (93/62)	11.5	29
	NGPU-LM	55.2 (97/39)	12.5	1807
	GPU-PB$_{uw}$	75.0 (94/62)	11.5	1784
	GPU-PB	**83.2** (90/77)	**10.2**	**1777**
递归神经网络转录器				
贪婪	–	42.5 (96/27)	12.8	1832
	CTC-WS	**80.0** (90/72)	**10.1**	639
	NGPU-LM	68.5 (96/54)	10.8	1812
	GPU-PB$_{uw}$	65.0 (94/50)	12.1	1759
	GPU-PB	70.4 (92/57)	10.7	1753
光束	–	44.2 (97/29)	12.8	1467
	NGPU-LM	58.6 (97/42)	12.0	1426
	GPU-PB$_{uw}$	80.9 (93/72)	10.1	1417
	GPU-PB	**82.9** (90/76)	**9.6**	**1430**