使用统一的文本到文本 Transformer 探索迁移学习的局限性

科林·拉菲] *

诺姆沙泽尔*

亚当·罗伯茨*

凯瑟琳·李*

沙兰纳朗

迈克尔·马特纳

Yanqi Zhou

Wei Li

彼得·刘

谷歌，山景城，CA 94043，美国 CRAFFEL@GMAIL.COM

NOAM@GOOGLE.COM

ADAROB@GOOGLE.COM

KATHERINELEE@GOOGLE.COM

SHARANNARANG@GOOGLE.COM

MMATENA@GOOGLE.COM

YANQIZ@GOOGLE.COM

MWEILI@GOOGLE.COM

PETERJLIU@GOOGLE.COM

编辑：伊万·蒂托夫

Abstract

迁移学习，其中模型首先在数据丰富的任务上进行预训练，然后在下游任务上进行微调，已成为自然语言处理 (NLP) 中的一项强大技术。迁移学习的有效性催生了多种方法、方法和实践。在本文中，我们通过引入一个将所有基于文本的语言问题转换为文本到文本格式的统一框架，探索了 NLP 迁移学习技术的前景。我们的系统研究比较了数十种语言理解任务的预训练目标、架构、未标记数据集、迁移方法和其他因素。通过将我们探索的见解与规模和我们新的“Colossal Clean Crawled Corpus”相结合，我们在许多基准测试中取得了最先进的结果，包括摘要、问答、文本分类等。为了促进 NLP 迁移学习的未来工作，我们发布了我们的数据集、预训练模型和代码。 [1]

关键词：迁移学习，自然语言处理，多任务学习，注意力模型，深度学习

1. 介绍

训练机器学习模型以执行自然语言处理 (NLP) 任务通常需要该模型能够以适合下游学习的方式处理文本。这可以粗略地视为开发允许模型"理解"文本的通用知识。这种知识的范围可以从低级（例如拼写或词义）到高级（例如，大多数背包都装不下大号号角（形容一个物体的尺寸或体积超过了常规限制，或者用来比喻某个概念、计划或想法过于庞大或不切实际））。在现代机器学习实践中，很少明确地提供这些知识；相反，它通常是作为辅助任务的一部分来学习的。例如，历史上常见的方法是使用词向量 (Mikolov et al., 2013b a; Pennington et al., 2014) 将词标识映射到连续表示，理想情况下，相似的词映射到相似的向量。这些向量通常是通过一个目标来学习的，例如，鼓励将同时出现的词放置在连续空间附近（Mikolov 等人，2013b）。

最近，在数据丰富的任务上预训练整个模型变得越来越普遍。理想情况下，这种预训练会使模型发展出通用的能力和知识，然后可以将这些能力和知识转移到下游任务中。在迁移学习到计算机视觉的应用中（Oquab et al., 2014; Jia et al., 2014; Huh et al., 2016; Yosinski et al., 2014），预训练通常通过监督学习完成标记数据集，如 ImageNet（Russakovsky 等人，2015 年；Deng 等人，2009 年）。相比之下，NLP 中用于迁移学习的现代技术通常使用无监督学习对未标记数据进行预训练。这种方法最近已被用于在许多最常见的 NLP 基准测试中获得最先进的结果（Devlin 等人，2018 年；Yang 等人，2019 年；Dong 等人，2019 年；Liu 等人。，2019c；Lan 等人，2019）。除了经验优势之外，NLP 的无监督预训练特别有吸引力，因为由于互联网，未标记的文本数据可以大量获得——例如，Common Crawl 项目 [2] 每月产生大约 20TB 的从网页中提取的文本数据。这非常适合神经网络，它已被证明具有显着的可扩展性，即通常可以通过在更大的数据集上训练更大的模型来获得更好的性能（Hestness 等人，2017 年；Shazeer 等人，2017 年）。 ，2017 年；Jozefowicz 等人，2016 年；Mahajan 等人，2018 年；Radford 等人，2019 年；Shazeer 等人，2018 年；Huang 等人，2018b；Keskar 等人，2019a)。

这种协同作用导致了最近为 NLP 开发迁移学习方法的大量工作，产生了广泛的预训练目标（Howard 和 Ruder，2018；Devlin 等，2018；Yang 等，2019； Dong 等人，2019 年），未标记数据集（Yang 等人，2019 年；Liu 等人，2019c；Zellers 等人，2019 年），基准（Wang 等人，2019b，2018 年；Conneau 和 Kiela，2018 年））、fine-tuning 方法（Howard 和 Ruder，2018 年；Houlsby 等人，2019 年；Peters 等人，2019 年）等。这个新兴领域的快速进步和技术多样性使得比较不同的算法、梳理新贡献的效果以及理解现有迁移学习方法的空间变得困难。由于需要更严格的理解，我们利用统一的方法进行迁移学习，使我们能够系统地研究不同的方法并突破该领域的当前极限。

我们工作的基本思想是将每个文本处理问题视为"文本到文本"问题，即将文本作为输入并生成新文本作为输出。这种方法受到之前 NLP 任务统一框架的启发，包括将所有文本问题转换为问答（McCann 等人，2018 年）、语言建模（Radford 等人，2019 年）或跨度提取 Keskar 等人(2019b) 任务。至关重要的是，文本到文本框架允许我们将相同的模型、目标、训练过程和解码过程直接应用于我们考虑的每项任务。我们通过评估各种基于英语的 NLP 问题的性能来利用这种灵活性，包括问答、文档

2. http://commoncrawl.org

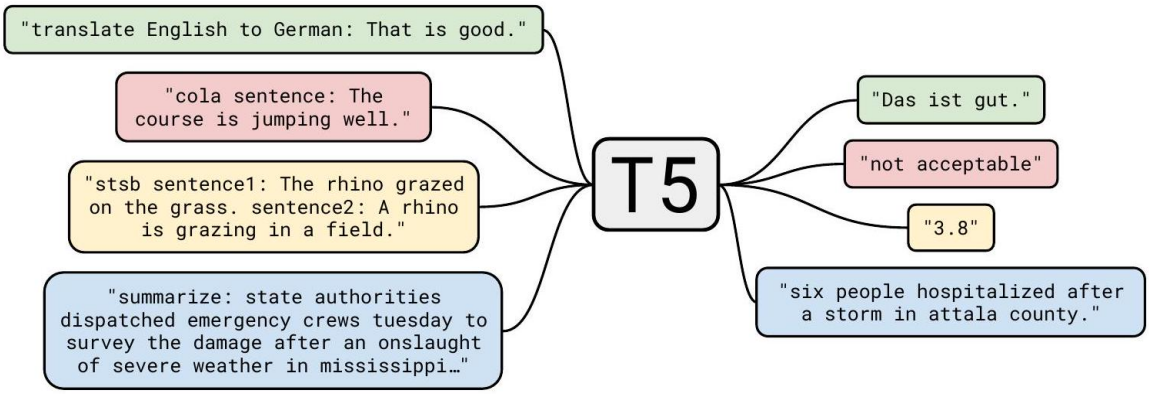

图 1：我们的文本到文本框架图。我们考虑的每项任务——包括翻译、问答和分类——都被视为将我们的模型文本作为输入并对其进行训练以生成一些目标文本。这允许我们在不同的任务集中使用相同的模型、损失函数、超参数等。它还为我们的实证调查中包含的方法提供了一个标准测试平台。“T5”指的是我们的模型，我们称之为“文本到文本转换的 Transformer（**T**ext-**to**-**T**ext **T**ransfer **T**ransformer）”。

总结和情感分类，仅举几例。通过这种统一的方法，我们可以比较不同迁移学习目标、未标记数据集和其他因素的有效性，同时通过扩展模型和数据集来探索 NLP 迁移学习的局限性，超出之前的考虑范围。

我们强调，我们的目标不是提出新方法，而是提供有关该领域现状的全面视角。因此，我们的工作主要包括对现有技术的调查、探索和经验比较。我们还通过扩大我们系统研究（训练模型多达 110 亿个参数）的见解来探索当前方法的局限性，以便在我们考虑的许多任务中获得最先进的结果。为了进行这种规模的实验，我们引入了“Colossal Clean Crawled Corpus”(C4)，这是一个由从网络上抓取的数百 GB 干净英文文本组成的数据集。认识到迁移学习的主要效用是在数据稀缺环境中利用预训练模型的可能性，我们发布了我们的代码、数据集和预训练模型。 [1]

本文的其余部分结构如下：在下一节中，我们将讨论我们的基本模型及其实现、将每个文本处理问题制定为文本到文本任务的过程，以及我们考虑的任务套件。在第 3 节中，我们展示了大量探索 NLP 迁移学习领域的实验。在本节的末尾（第 3.7 节），我们结合了系统研究的见解，以在各种基准测试中获得最先进的结果。最后，我们总结了我们的结果，并在第 4 节中总结了对未来的展望。

2. 设置

在展示我们大规模实证研究的结果之前，我们回顾了理解我们的结果所需的必要背景主题，包括 Transformer 模型架构和我们评估的下游任务。我们还介绍了我们将每个问题视为文本到文本任务的方法，并描述了我们的“Colossal Clean Crawled Corpus”(C4)，这是我们创建的基于 Common Crawl 的数据集，作为未标记文本数据的来源。我们将我们的模型和框架称为“文本到文本转换 Transformer”(**T5**)。

2.1. 模型

NLP 迁移学习的早期结果利用了递归神经网络（Peters 等人，2018 年；Howard 和 Ruder，2018 年），但最近使用基于“Transformer”架构的模型变得越来越普遍（Vaswani 等人，2018 年）。，2017). Transformer 最初被证明对机器翻译有效，但随后被广泛用于各种 NLP 设置中（Radford 等人，2018 年；Devlin 等人，2018 年；McCann 等人，2018 年；Yu等人，2018 年）。由于它越来越普遍，我们研究的所有模型都基于 Transformer 架构。除了下面提到的细节和我们在第 3.2 节中探讨的变体之外，我们并没有明显偏离最初提出的这种架构。我们没有提供该模型的全面定义，而是建议感兴趣的读者参阅原始论文（Vaswani 等人，2017 年）或后续教程 [3] 以获得更详细的介绍。

Transformer 的主要组成部分是自注意力（Cheng 等人，2016 年）。自注意力是注意力的一种变体 (Graves, 2013; Bahdanau et al., 2015)，它通过用序列其余部分的加权平均值替换每个元素来处理序列。最初的 Transformer 由编码器-解码器架构组成，旨在用于序列到序列（Sutskever 等人，2014 年；Kalchbrenner 等人，2014 年）任务。最近，使用由单个 Transformer 层堆栈组成的模型也变得很普遍，这些模型使用不同形式的自注意力来生成适合语言建模的架构（Radford 等人，2018 年；Al-Rfou 等人，2019 年）) 或分类和跨度预测任务 (Devlin et al., 2018; Yang et al., 2019)。我们在第 3.2 节中凭经验探索了这些架构变体。

总的来说，我们的编码器-解码器 Transformer 实现严格遵循其最初提出的形式 (Vaswani et al., 2017)。首先，tokens 的输入序列被映射到 embeddings 的序列，然后将其传递给编码器。编码器由一堆“块”组成，每个“块”包含两个子组件：一个自注意力层，后跟一个小型前馈网络。层归一化 (Ba et al., 2016) 应用于每个子组件的输入。我们使用层归一化的简化版本，其中仅重新缩放激活并且不应用附加偏差。在层归一化之后，残差跳跃连接 (He et al., 2016) 将每个子组件的输入添加到其输出。Dropout (Srivastava et al., 2014) 应用于前馈网络、跳跃连接、注意力权重以及整个堆栈的输入和输出。解码器在结构上与编码器相似，除了它包含一个标准的，每个自注意力层之后关注编码器输出的注意力机制。解码器中的自注意力机制也使用了一种自回归或因果自注意力的形式，它只允许模型关注过去的输出。最终解码器块的输出被馈送到具有 softmax 输出的密集层，其权重与输入 embedding 矩阵共享。 Transformer 中的所有注意力机制都被分成独立的“头”，其输出在被进一步处理之前被 concat 启用。

由于自注意力是顺序无关的（即它是对集合的操作），因此通常向 Transformer 提供明确的位置信号。虽然最初的 Transformer 使用正弦位置信号或学习位置 embeddings，但最近使用相对位置 embeddings 变得越来越普遍（Shaw 等人，2018 年；Huang 等人，2018a）。相对位置 embedding 不是为每个位置使用固定的 embeddings，而是根据在自我注意机制中比较的“键”和“查询”之间的偏移量产生不同的学习 embedding。我们使用位置 embeddings 的简化形式，其中每个“embedding” 只是一个标量，添加到用于计算注意力权重的相应 logit。为了提高效率，我们还在模型的所有层中共享位置 embedding 参数，尽管在给定层内每个注意力头使用不同的学习位置 embedding。通常，学习固定数量的 embeddings，每个对应于一系列可能的键查询偏移量。在这项工作中，我们对所有模型使用 32 embeddings，其范围按对数方式增加到 128 的偏移量，超过 128，我们将所有相对位置分配给相同的 embedding。请注意，给定层对超过 128 tokens 的相对位置不敏感，但后续层可以通过结合前一层的局部信息来建立对更大偏移的敏感性。总而言之，我们的模型大致相当于 Vaswani 等人提出的原始 Transformer(2017) 除了移除 Layer Norm 偏差，将层归一化放置在残差路径之外，并使用不同的位置 embedding 方案。由于这些架构变化与我们在迁移学习实证调查中考虑的实验因素正交，我们将消除它们对未来工作的影响。

作为我们研究的一部分，我们试验了这些模型的可扩展性，即当它们具有更多参数或层时，它们的性能如何变化。训练大型模型可能并非易事，因为它们可能不适合单台机器并且需要大量计算。因此，我们结合使用模型和数据并行性，并在 Cloud TPU Pod 的“切片”上训练模型。 [5] TPU pod 是多机架 ML 超级计算机，包含 1,024 个 TPU v3 芯片，通过高速 2D 网状互连与支持的 CPU 主机连接。我们利用 Mesh TensorFlow 库（Shazeer 等人，2018 年）来简化模型并行性和数据并行性的实施（Krizhevsky，2014 年）。

2.2. 巨大干净的爬取语料库

之前关于 NLP 迁移学习的大部分工作都使用大型未标记数据集进行无监督学习。在本文中，我们感兴趣的是测量这种未标记数据的质量、特征和大小的影响。为了生成满足我们需求的数据集，我们利用 Common Crawl 作为从网络上抓取的文本来源。

Common Crawl 以前被用作 NLP 的文本数据源，例如训练一个 n-gram 语言模型（Buck et al., 2014），作为训练数据用于常识推理（Trinh 和 Le，2018），用于挖掘机器翻译的并行文本（Smith 等，2013），作为预训练数据集（Grave 等，2018；Zellers 等，2019；Liu et al., 2019c)，甚至只是作为用于测试优化器的巨型文本语料库 (Anil et al., 2019)。

Common Crawl 是一个公开可用的网络存档，它通过从抓取的 HTML 文件中删除标记和其他非文本内容来提供“网络提取文本”。这个过程每月产生大约 20TB 的抓取文本数据。不幸的是，大部分生成的文本不是自然语言。相反，它主要包含乱码或样板文本，如菜单、错误消息或重复文本。此外，大量被抓取的文本包含不太可能对我们考虑的任何任务有帮助的内容（攻击性语言、占位符文本、源代码等）。为了解决这些问题，我们使用了以下启发式方法来清理 Common Crawl 的网络提取文本：

- 我们只保留以结尾标点符号（即句号、感叹号、问号或结束引号）结尾的行。
- 我们丢弃了任何少于 5 个句子的页面，只保留至少包含 3 个单词的行。
- 我们删除了任何包含“肮脏、顽皮、淫秽或其他不良词语列表”中任何词语的页面。 ′
- 许多被抓取的页面包含警告，说明应该启用 Javascript，因此我们删除了任何包含 Javascript 一词的行。
- 一些页面有占位符“lorem ipsum”文本；我们删除了所有出现“lorem ipsum”短语的页面。
- 某些页面无意中包含了代码。由于大括号 "{ { 出现在许多编程语言中（例如在网络上广泛使用的 Javascript）而不是自然文本，我们删除了所有包含大括号的页面。
- 在数据集中进行数据去重操作时，我们丢弃了重复出现超过一次的所有三句话的片段，只保留其中的一份。

此外，由于我们的大部分下游任务都集中在英语文本上，因此我们使用 langdetect [7] 以至少 0.99 的概率过滤掉任何未分类为英语的页面。我们的启发式方法受到过去使用 Common Crawl 作为 NLP 数据源的工作的启发：例如，Grave 等人(2018) 还使用自动语言检测器过滤文本并丢弃短行，Smith 等人 (2013);坟墓等(2018) 都执行行级重复数据删除。然而，我们选择创建一个新的数据集，因为之前的数据集使用了一组更有限的过滤启发式方法，不公开可用，和/或范围不同（例如，仅限于新闻数据（Zellers 等人，2019 年； Liu et al., 2019c)，仅包含 Creative Commons 内容（Habernal et al., 2016），或专注于机器翻译的并行训练数据（Smith et al., 2013））。

为了组装我们的基础数据集，我们下载了 2019 年 4 月的网络提取文本并应用了上述过滤。这产生的文本集合不仅比大多数用于预训练的数据集（约 750 GB）大几个数量级，而且还包含相当干净和自然的英文文本。我们将这个数据集称为“Colossal Clean Crawled Corpus”（或简称 C4），并将其作为 TensorFlow Datasets 的一部分发布。 [8] 我们在第 3.4 节中考虑了使用此数据集的各种替代版本的影响。

2.3. 下游任务

我们在本文中的目标是衡量一般语言学习能力。因此，我们研究了各种基准的下游性能，包括机器翻译、问答、抽象摘要和文本分类。具体来说，我们测量了 GLUE 和 SuperGLUE 文本分类元基准的性能； CNN/每日邮报摘要摘要； SQuAD问答；和 WMT 英语到德语、法语和罗马尼亚语的翻译。所有数据均来自 TensorFlow 数据集。 [9]

GLUE (Wang et al., 2018) 和 SuperGLUE (Wang et al., 2019b) 各自包含一组旨在测试一般语言理解能力的文本分类任务：

- 句子可接受性判断（CoLA (Warstadt et al., 2018))
- 情绪分析 (SST-2 (Socher et al., 2013))
- 释义/句子相似性（MRPC（Dolan 和 Brockett，2005 年）、STS-B（Cer 等人，2017 年）、QQP（Iyer 等人，2017 年））
- 自然语言推理（MNLI（Williams 等人，2017 年）、QNLI（Rajpurkar 等人，2016 年）、RTE（Dagan 等人，2005 年）、CB（De Marneff 等人，2019 年））
- 共指消解 (WNLI 和 WSC (Levesque et al., 2012))
- 句子补全（COPA（Roemmele 等人，2011 年））
- 词义消歧（WIC（Pilehvar 和 Camacho-Collados，2018））
- 问答（MultiRC（Khashabi 等人，2018 年）、ReCoRD（Zhang 等人，2018 年）、BoolQ（Clark 等人，2019 年））

我们使用由 GLUE 和 SuperGLUE 基准发布的数据集。为简单起见，当 fine-tuning 时，我们通过 concat 启用所有组成数据集，将 GLUE 基准测试中的所有任务（以及类似的 SuperGLUE）视为单个任务。正如 Kocijan 等人所建议的那样(2019) 我们还在组合 SuperGLUE 任务中包含了定代词解析 (DPR) 数据集（Rahman 和 Ng，2012）。

CNN/Daily Mail (Hermann et al., 2015) 数据集作为问答任务引入，但被 Nallapati 等人改编为文本摘要(2016);我们使用 See 等人的非匿名版本（2017）作为抽象摘要任务。 SQuAD (Rajpurkar et al., 2016) 是一种常见的问答基准。在我们的实验中，模型被输入问题及其上下文，并被要求生成答案 token-by-token。对于 WMT English to German，我们使用与 (Vaswani et al., 2017) 相同的训练数据（即 News Commentary v13、Common Crawl、Europarl v7）和 newstest2013 作为验证集（Bojar et al., 2014）。对于英语到法语，我们使用 2015 年的标准训练数据和 newstest2014 作为验证集 (Bojar et al., 2015)。对于英语到罗马尼亚语（标准的低资源机器翻译基准），我们使用 WMT 2016 的训练集和验证集（Bojar 等人，2016 年）。请注意，我们只对英语数据进行预训练，因此为了学习翻译给定模型，需要学习以新语言生成文本。

2.4. 输入和输出格式

为了在上述不同的任务集上训练一个单一的模型，我们将我们考虑的所有任务转换为“文本到文本”的格式——也就是说，在这个任务中，模型被输入一些文本作为上下文或调节然后被要求产生一些输出文本。该框架为预训练和 fine-tuning 提供了一致的训练目标。具体来说，无论任务如何，该模型都以最大似然目标（使用“教师强制（teacher forcing）”（Williams 和 Zipser，1989））进行训练。为了指定模型应该执行的任务，我们在将原始输入序列提供给模型之前向其添加特定于任务的（文本）前缀。

例如，要求模型翻译句子“That is good”。从英语到德语，模型将被输入“translate English to German: That is good.”的序列。并将被训练输出“Das ist gut”。对于文本分类任务，该模型简单地预测与目标标签对应的单个词。例如，在 MNLI 基准测试（Williams 等人，2017 年）上，目标是预测前提是否暗示（“蕴涵（entailment）”）、矛盾（“矛盾（contradiction）”）或两者都不（“中性（neutral）”）假设。通过我们的预处理，输入序列变成“mnli premise: I hate pigeons. hypothesis: My feelings towards pigeons are filled with animosity.”相应的目标词是“entailment”。请注意，如果我们的模型在文本分类任务上输出的文本与任何可能的标签都不对应（例如，如果模型在任务的唯一可能标签为“蕴含”、“中性”、或“矛盾”时输出“汉堡包”，则会出现问题”）。在这种情况下，我们总是认为模型的输出是错误的，尽管我们从未在任何经过训练的模型中观察到这种行为。请注意，用于给定任务的文本前缀的选择本质上是一个超参数；我们发现改变前缀的确切措辞影响有限，因此没有对不同的前缀选择进行广泛的实验。图 1 显示了我们的文本到文本框架的图表以及一些输入/输出样本。我们在附录 D 中为我们研究的每个任务提供了预处理输入的完整样本。

我们的文本到文本框架遵循之前将多个 NLP 任务转换为通用格式的工作：McCann 等人(2018) 提出了“自然语言十项全能”，这是一个基准，它对一组十项 NLP 任务使用一致的问答格式。自然语言十项全能还规定所有模型必须是多任务的，即能够同时处理所有任务。相反，我们允许单独 fine-tuning 每个单独任务的模型，并使用短任务前缀而不是明确的问答格式。雷德福等人(2019) 通过将一些输入作为前缀提供给模型，然后对输出进行自回归采样来评估语言模型的 zero-shot 学习能力。例如，自动摘要是通过输入文档后跟文本“TL;DR:”（“too long, didn't read”的缩写，常见缩写）来完成的，然后通过自回归解码预测摘要。我们主要考虑在使用单独的解码器生成输出之前使用编码器显式处理输入的模型，并且我们专注于迁移学习而不是 zero-shot 学习。最后，Keskar 等人(2019b) 将许多 NLP 任务统一为“跨度提取”，其中将与可能的输出选择对应的文本附加到输入中，并训练模型以提取与正确选择对应的输入跨度。相比之下，我们的框架还允许生成任务，如机器翻译和抽象摘要，在这些任务中无法枚举所有可能的输出选择。

我们能够直接将我们考虑的所有任务转换为文本到文本的格式，STS-B 除外，这是一个回归任务，其目标是预测 1 和 5 之间的相似性得分。我们发现这些分数中的大多数都以 0.2 的增量进行注释，因此我们简单地将任何分数四舍五入到最接近的增量 0.2 并将结果转换为数字的文字字符串表示形式（例

如，浮点值 2.57 将映射到字符串 " 2.6 ”)。在测试时，如果模型输出的字符串对应于 1 到 5 之间的数字，我们将其转换为浮点值；否则，我们将模型的预测视为不正确。这有效地重新表示了STS-B 回归问题作为 21 类分类问题。

另外，我们还将 Winograd 任务（来自 GLUE 的 WNLI、来自 SuperGLUE 的 WSC 以及我们添加到 SuperGLUE 的 DPR 数据集）转换为更适合文本到文本框架的更简单格式。来自 Winograd 任务的样本包括一段包含歧义代词的文本段落，该代词可能指代段落中的多个名词短语。例如，文章可能是“The city councilmen refused the demonstrators a permit because they fear violence.”，其中包含歧义代词“they”，可以指代“city councilmen”或“demonstrators”。我们通过突出显示文本段落中的歧义代词并要求模型预测它所指的名词，将 WNLI、WSC 和 DPR 任务视为文本到文本的问题。上面提到的样本将转换为输入“The city councilmen refused the demonstrators a permit because *they* feared violence。”并且该模型将被训练以预测目标文本“The city councilmen”。

对于 WSC，样本包含段落、歧义代词、候选名词和反映候选是否与代词匹配的真/假标签（忽略任何冠词）。我们只训练带有“True”标签的样本，因为我们不知道带有“False”标签的样本的正确名词目标。对于评估，如果模型输出中的单词是候选名词短语中的单词的子集（反之亦然），我们将分配一个“True”标签，否则分配一个“False”标签。这删除了大约一半的 WSC 训练集，但 DPR 数据集增加了大约 1,000 个代词解析样本。来自 DPR 的样本使用正确的指称名词进行注释，使得以上面列出的格式使用此数据集变得容易。

WNLI 训练集和验证集与 WSC 训练集有很大的重叠。为了避免将验证样本泄漏到我们的训练数据中（第 3.5.2 节的多任务实验中的一个特殊问题），我们因此从不在 WNLI 上训练，也从不报告 WNLI 验证集上的结果。省略 WNLI 验证集上的结果是标准做法（Devlin 等人，2018 年），因为它相对于训练集是“对抗性的”，即验证样本都是训练样本的轻微扰动版本，而相反标签。因此，每当我们报告验证集时，我们都不将 WNLI 包括在平均 GLUE 分数中（除第 3.7 节外的所有部分，其中结果显示在测试集上）。将样本从 WNLI 转换为上述“指称名词预测”变体有点复杂；我们在附录 B 中描述了这个过程

3. 实验

NLP 迁移学习的最新进展来自各种各样的发展，例如新的预训练目标、模型架构、未标记的数据集等。在本节中，我们对这些技术进行了实证调查，希望梳理出它们的贡献和意义。然后，我们将获得的见解结合起来，以在我们考虑的许多任务中实现最先进的水平。由于 NLP 的迁移学习是一个快速发展的研究领域，我们不可能在实证研究中涵盖所有可能的技术或想法。对于更广泛的文献综述，我们推荐 Ruder 等人最近的一项调查（2019）。

我们通过采用合理的baseline（在第 3.1 节中描述）并一次更改设置的一个方面来系统地研究这些贡献。例如，在第 3.3 节中，我们测量不同无监督目标的性能，同时保持我们实验的其余部分 pipeline 固定。这种“坐标上升”方法可能会错过二阶效应（例如，某些特定的无监督目标可能在比我们的 baseline设置更大的模型上效果最好），但对我们研究中的所有因素进行组合探索将非常昂贵.在未来的工作中，我们希望更彻底地考虑我们研究的方法的组合会富有成效。

我们的目标是比较针对不同任务集的各种不同方法，同时尽可能多地固定因素。为了满足这个目标，在某些情况下我们并不完全复制现有的方法。例如，像 BERT（Devlin 等人，2018 年）这样的“仅编码器”模型旨在为每个输入 token 生成单个预测或对整个输入序列生成单个预测。这使得它们适用于分类或跨度预测任务，但不适用于翻译或抽象摘要等生成任务。因此，我们考虑的模型架构均不与 BERT 相同或由仅编码器结构组成。相反，我们测试了本质上相似的方法——例如，我们在第 3.3 节中考虑了与 BERT 的“masked 语言建模”目标类似的目标，并且我们在第 3.2 节中考虑了在文本分类任务上与 BERT 行为相似的模型架构.

在以下小节中概述了我们的baseline实验设置之后，我们对模型架构（第 3.2 节）、无监督目标（第 3.3 节）、预训练数据集（第 3.4 节）、迁移方法（第 3.5 节）和缩放进行了实证比较（第 3.6 节）。在本节的最后，我们将研究的见解与规模相结合，以在我们考虑的许多任务中获得最先进的结果（第 3.7 节）。

3.1. baseline

我们的baseline目标是反映典型的现代实践。我们使用简单的去噪目标预训练标准 Transformer（在第 2.1 节中描述），然后在我们的每个下游任务上分别 fine-tune。我们在以下小节中描述了此实验设置的详细信息。

3.1.1. 模型

对于我们的模型，我们使用 Vaswani 等人提出的标准编码器-解码器 Transformer(2017)。虽然许多用于 NLP 的迁移学习的现代方法使用仅由单个“堆栈”组成的 Transformer 架构（例如用于语言建模（Radford 等人，2018 年；Dong 等人，2019 年）或分类和跨度预测（Devlin et al., 2018; Yang et al., 2019)），我们发现使用标准的编码器-解码器结构在生成和分类任务上都取得了很好的效果。我们在 3.2 节中探讨了不同模型架构的性能。

我们的baseline模型旨在使编码器和解码器在大小和配置上都与“BERT BASE”（Devlin 等人，2018 年）堆栈相似。具体来说，编码器和解码器都由 12 个块组成（每个块包含自注意力、可选的编码器-解码器注意力和前馈网络）。每个块中的前馈网络由一个输出维数为 $d_{\text{ff}} = 3072$ 的密集层组成，后跟 ReLU 非线性和另一个密集层。所有注意力机制的“键”和“值”矩阵的内维都是 $d_{\text{kv}} = 64$，所有注意力机制都有12个头。所有其他子层和 embeddings 的维度为 $d_{\text{model}} = 768$。总的来说，这会产生一个具有大约 2.2 亿个参数的模型。这大约是 BERT BASE 参数数量的两倍，因为我们的基线模型包含两个层堆栈而不是一个。。对于正则化，我们在模型中应用 dropout 的任何地方都使用 0.1 的 dropout 概率。

3.1.2. 训练

如第 2.4 节所述，所有任务都被表述为文本到文本任务。这使我们能够始终使用标准最大似然进行训练，即使用教师强制（Williams 和 Zipser，1989）和交叉熵损失。优化器，我们使用 AdaFactor（Shazeer 和 Stern，2018）。在测试时，我们使用贪心解码（即在每个时间步选择最高概率的 logit）。

我们在 fine-tuning 之前针对 C4 上的 $2^{19} = 524,288$ 步骤对每个模型进行预训练。我们使用 512 的最大序列长度和 128 序列的 batch 大小。只要有可能，我们就将多个序列“打包”到 batch 的每个entity 中，以便我们的 batches 大致包含 $2^{16} = 65,536$ tokens。总的来说，这个 batch 大小和步数对应于 $2^{35} \approx 34\ \text{B}$ tokens 上的预训练。这比使用大约 137B tokens 的 BERT（Devlin 等人，2018 年）或使用大约 $2.2\ \text{T}$ tokens 的 RoBERTa（Liu 等人，2019c）要少得多。仅使用 2^{35} tokens 会产生合理的计算预算，同时仍然提供足够数量的预训练以获得可接受的性能。我们考虑 3.6和3.7节更多步骤的预训练效果。请注意， 2^{35} tokens 仅涵盖整个 C4 数据集的一小部分，因此我们在预训练期间从不重复任何数据。

在预训练期间，我们使用“平方根倒数”学习率 schedule: $1/\sqrt{\max(n, k)}$ 其中 n 是当前训练迭代， k 是预热步骤数（设置为 10^4 在我们所有的实验）。这为第一个 10^4 步骤设置了 0.01 的恒定学习率，然后以指数方式衰减学习率，直到预训练结束。我们还尝试使用三角学习率（Howard 和 Ruder，2018），这产生了稍微好一点的结果，但需要提前知道训练步骤的总数。由于我们将在某些实验中改变训练步骤的数量，因此我们选择更通用的平方根倒数 schedule。

我们的模型在所有任务上的 $2^{18} = 262,144$ 步骤都是 fine-tuned。选择此值作为高资源任务（即具有大数据集的任务）和低资源任务（较小数据集）之间的权衡，前者受益于额外的 fine-tuning，后者很快过拟合。在 fine-tuning 期间，我们继续使用具有 128 个长度为 512 的序列的 batches（即 2^{16} tokens

每个 batch）。当 fine-tuning 时，我们使用 0.001 的恒定学习率。我们每 5,000 步保存一个 checkpoint，并报告与最高验证性能相对应的模型 checkpoint 的结果。对于多个任务的模型 fine-tuned，我们独立地为每个任务选择最好的 checkpoint。对于除第 3.7 节中的实验之外的所有实验，我们在验证集中报告结果以避免在测试集上执行模型选择。

3.1.3. 词汇

我们使用 SentencePiece（Kudo 和 Richardson，2018 年）将文本编码为 WordPiece tokens（Sennrich 等人，2015 年；Kudo，2018 年）。对于所有实验，我们使用 32,000 个单词的词汇表。由于我们最终 fine-tune 我们的英语到德语、法语和罗马尼亚语翻译模型，我们还要求我们的词汇涵盖这些非英语语言。为了解决这个问题，我们将 C4 中使用的 Common Crawl 抓取的页面分类为德语、法语和罗马尼亚语。然后，我们在 10 部分英语 C4 数据与 1 部分分类为德语、法语或罗马尼亚语的数据的混合上训练了我们的 SentencePiece 模型。这个词汇表在我们模型的输入和输出中共享。请注意，我们的词汇表使得我们的模型只能处理一组预先确定的固定语言。

3.1.4. 无监督目标

利用未标记的数据来预训练我们的模型需要一个不需要标签但（不严谨地说）能教会模型可概括的知识的目标，这些知识将在下游任务中有用。将预训练的迁移学习范式和 fine-tuning 模型的所有参数应用于 NLP 问题的初步工作使用因果语言建模目标进行预训练（Dai 和 Le，2015；Peters 等，2018；Radford 等等人，2018 年；霍华德和罗德，2018 年）。然而，最近表明“去噪”目标（Devlin 等人，2018 年；Taylor，1953 年）（也称为“masked 语言建模”）可以产生更好的性能，因此它们很快成为标准。在去噪目标中，模型经过训练以预测输入中缺失或破坏的 tokens。

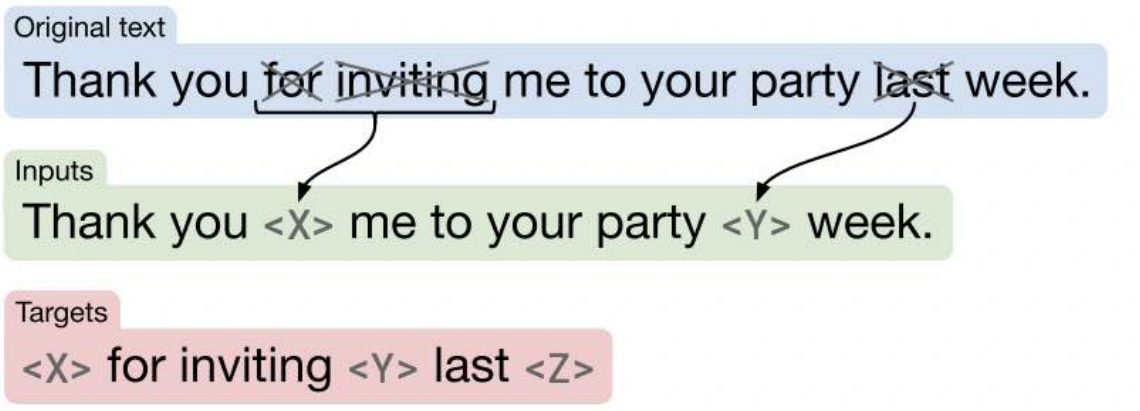

图 2：我们在baseline模型中使用的目标示意图。在此样本中，我们处理句子“Thank you for inviting me to your party last week.(感谢您上周邀请我参加您的聚会。)”随机选择单词“for”、“inviting”和“last”（标有 × ）用于破坏。破坏的 tokens 的每个连续跨度都被样本中唯一的标记 token （显示为 $\langle X \rangle$ 和 $\langle Y \rangle$ ）替换。由于“for”和“inviting”连续出现，因此它们被单个哨兵 $\langle \mathrm{X} \rangle$ 取代。然后输出序列由丢弃的跨度组成，由用于在输入中替换它们的哨兵 tokens 加上最终哨兵 token $\langle \mathrm{Z} \rangle$ 分隔。

灵感来自 BERT 的“masked 语言建模”目标和“word dropout”正则化技术（Bowman et al., 2015），我们设计了一个随机抽样的目标，然后在输入序列中丢弃 tokens 的 15%。所有连续的辍学 tokens 跨度都被单个哨兵 token 取代。每个哨兵 token 都分配有一个序列唯一的 token ID。哨兵 ID 是特殊的 tokens，它被添加到我们的词汇表中，不对应于任何词块。然后目标对应于 tokens 的所有退出范围，由输入序列中使用的相同标记 tokens 加上标记目标序列结尾的最终标记 token 分隔。我们选择 mask 连续跨度 tokens 和仅预测丢失 tokens 是为了降低预训练的计算成本。我们对第 3.3 节中的预训练目标进行了彻底调查。图 2 显示了应用此目标所产生的转换样本。我们根据经验将此目标与第 3.3 节中的许多其他变体进行了比较

3.1.5. 基准性能

在本节中，我们使用上述baseline实验程序展示结果，以了解在我们的下游任务套件中期望什么样的性能。理想情况下，我们会多次重复研究中的每个实验，以获得结果的置信区间。不幸的是，由于我们进行了大量实验，这将非常昂贵。作为更便宜的替代方案，我们从头开始训练我们的baseline模型 10 次（即使用不同的随机初始化和数据集改组），并假设基本模型的这些运行的方差也适用于每个实验变体。我们预计我们所做的大部分更改不会对运行间方差产生显着影响，因此这应该提供不同更改重要性的合理指示。另外，我们还测量了在没有预训练的情况下针对所有下游任务训练我们模型的 2^{18} 步骤（与我们用于 fine-tuning 的数字相同）的性能。这让我们了解在baseline设置中预训练对我们的模型有多大好处。

	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnR
* baseline平均值	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.6**
baseline标准偏差	0.235	0.065	0.343	0.416	0.112	0.090	0.10
无预训练	66.22	17.60	50.31	53.04	25.86	**39.77**	24.0

表 1：我们的baseline模型和训练程序取得的平均分数和标准差。为了进行比较，我们还报告了从头开始训练每项任务（即没有任何预训练）时用于 fine-tune baseline模型的相同步数的性能。此表中的所有分数（以及我们论文中除表 14 之外的每个表）均在每个数据集的验证集上报告。

在正文中报告结果时，我们只报告所有基准测试的一部分分数，以节省空间并便于解释。对于 GLUE 和 SuperGLUE，我们在“GLUE”和“SGLUE”标题下报告所有子任务的平均分数（按照官方基准的规定）。对于所有翻译任务，我们报告了 SacreBLEU v1.3.0（Post，2018）提供的 BLEU 分数（Papineni 等人，2002 年），具有“exp”平滑和“intl”token化。我们将 WMT 英语到德语、英语到法语和英语到罗马尼亚语的分数分别称为 EnDe、EnFr 和 EnRo。对于 CNN/每日邮报，我们发现模型在 ROUGE-1-F、ROUGE-2-F 和 ROUGE-L-F 指标（Lin，2004 年）上的性能高度相关，因此我们报告 ROUGE-2-F 分数单独在“CNNDM”标题下。同样，对于 SQuAD，我们发现“完全匹配”和“F1”分数的性能高度相关，因此我们单独报告“完全匹配”分数。我们在表 16 的附录 E 中提供了所有实验在每项任务上取得的每一个分数。

我们的结果表都经过格式化，因此每一行都对应于一个特定的实验配置，其中列给出了每个基准的分数。我们将在大多数表格中包含基准配置的平均性能。无论baseline配置出现在哪里，我们都会用 $\star$ 标记它（如表 1 的第一行）。我们还将粗体显示给定实验中最大值（最佳）的两个标准差范围内的任何分数。

我们的baseline结果如表 1 所示。总的来说，我们的结果与类似尺寸的现有模型相当。 例如，BERT $_{\text{BASE}}$ 在 SQuAD 上获得了 80.8 的精确匹配分数，在 MNLI 匹配上获得了 84.4 的准确率，而我们分别达到 80.88 和 84.24（见表 16）。请注意，我们无法直接将我们的baseline与 BERT $_{\text{BASE}}$ 进行比较，因为我们的模型是一个编码器-解码器模型，并且经过了大约 1/4 步数的预训练。毫不奇怪，我们发现预训练在几乎所有基准测试中都提供了显着的收益。唯一的例外是 WMT English to French，这是一个足够大的数据集，预训练的收益往往很小。我们将此任务包括在我们的实验中，以测试高资源制度下迁移学习的行为。由于我们通过选择性能最好的 checkpoint 来执行提前停止，我们的baseline和“无预训练”之间的巨大差异强调了预训练在数据有限的任务上提高了多少性能。虽然我们没有在本文中明确衡量数据效率的改进，但我们强调这是迁移学习范式的主要好处之一。

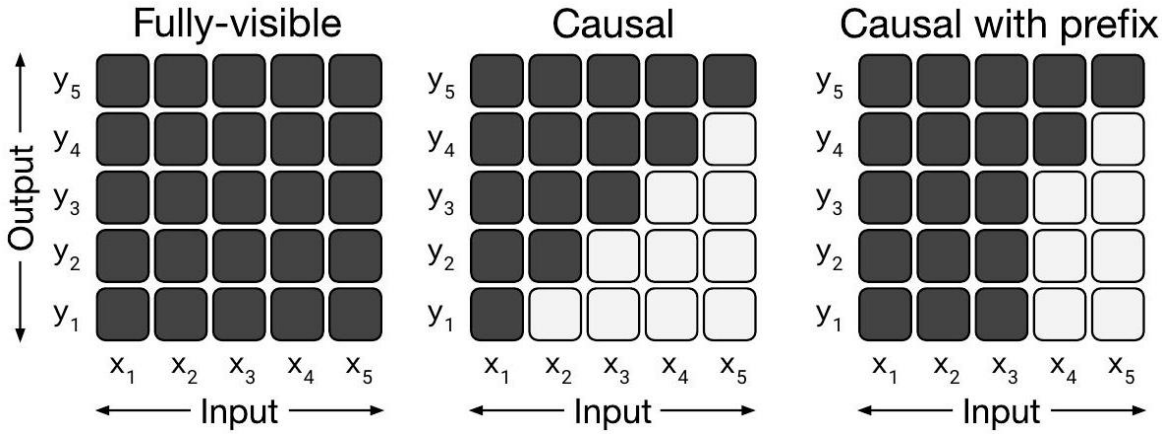

图 3：代表不同注意力 mask 模式的矩阵。自注意力机制的输入和输出分别表示为 x和 y。 i 行和 j 列的黑色单元格表示允许自注意机制在输出时间步长 j 处处理输入元素 i。浅色单元表示不允许自注意机制关注相应的 i 和 j 组合。左图：完全可见的 mask 允许自我注意机制在每个输出时间步处理完整输入。中间：因果 mask 阻止第 i 个输出元素依赖于"未来"的任何输入元素。右图：带有前缀的因果 masking 允许自我注意机制在输入序列的一部分上使用完全可见的 masking。

至于运行间方差，我们发现对于大多数任务，运行间的标准差小于任务baseline分数的 1%。此规则的例外情况包括 CoLA、CB 和 COPA，它们都是 GLUE 和 SuperGLUE 基准测试中的低资源任务。例如，在 CB 上，我们的baseline模型的平均 F1 分数为 91.22，标准差为 3.237（见表 16），这可能部分是由于 CB 的验证集仅包含 56 个样本。请注意，GLUE 和 SuperGLUE 分数计算为构成每个基准的任务的平均分数。因此，我们警告说，CoLA、CB 和 COPA 的高运行间方差会使单独使用 GLUE 和 SuperGLUE 分数比较模型变得更加困难。

3.2. 架构

虽然 Transformer 最初是通过编码器-解码器架构引入的，但许多关于 NLP 迁移学习的现代工作都使用了替代架构。在本节中，我们将回顾和比较这些架构变体。

3.2.1. 模型结构

不同架构的一个主要区别因素是模型中不同注意力机制使用的"mask"。回想一下 Transformer 中的自注意力操作将一个序列作为输入并输出一个相同长度的新序列。输出序列的每个entity都是通过计算输入序列entity的加权平均值生成的。具体来说，令 y_i 指输出序列的第 i 个元素， x_j 指输入序列的第 j 个entity。

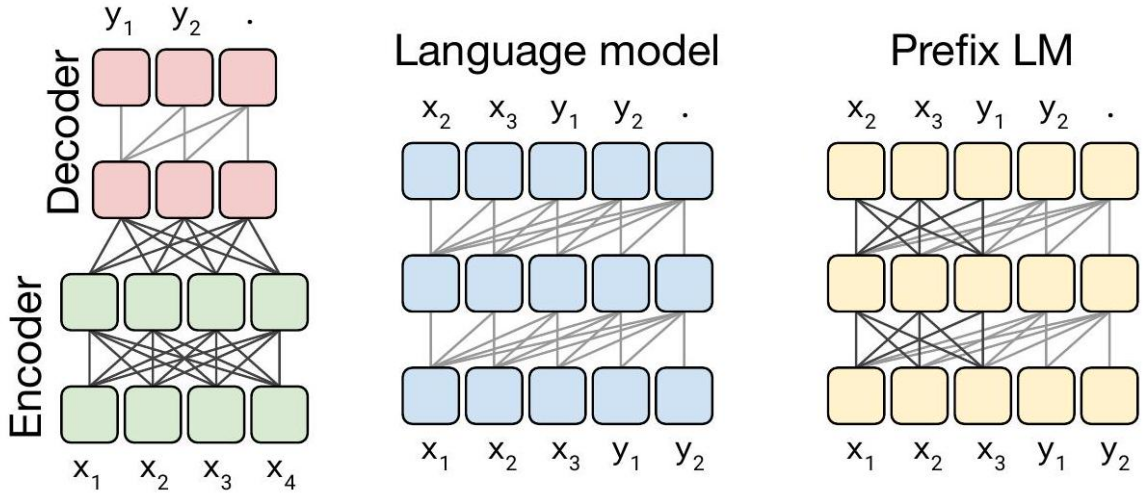

图 4：我们考虑的 Transformer 架构变体的示意图。在此图中，块表示序列的元素，线条表示注意力可见性。不同颜色的块组表示不同的 Transformer 层堆栈。深灰色线对应于完全可见的 masking，浅灰色线对应于因果 masking。我们用 "."表示一个特殊的序列结束 token，表示预测的结束。输入和输出序列分别表示为 x和 y。左图：标准的编码器-解码器架构在编码器和编码器-解码器

y_i 计算为 $\sum_j w_{i,j}x_j$，其中 $w_{i,j}$ 是由自注意力机制产生的标量权重，作为 x_i 和 x_j 的函数。然后使用注意力 mask 将某些权重归零，以限制在给定的输出时间步可以关注输入的哪些entity。我们将考虑的 mask 的图表如图 3 所示。例如，因果关系 mask（图 3，中间）将任何 $w_{i,j}$ 设置为零，如果 $j > i$。

我们考虑的第一个模型结构是编码器-解码器 Transformer，它由两层堆栈组成：输入序列的编码器和产生新输出序列的解码器。图 4 的左侧面板显示了此架构变体的示意图。

编码器使用"完全可见"的注意力 mask。完全可见的 masking 允许自注意力机制在生成其输出的每个entity时关注输入的任何entity。我们在图 3 左侧可视化了这种 masking 模式。这种形式的 masking 在关注"前缀"时是合适的，即提供给模型的一些上下文，稍后在进行预测时使用。 BERT（Devlin 等人，2018 年）还使用了完全可见的 masking 模式，并在输入中附加了一个特殊的"分类"token。 BERT 在对应于分类 token 的时间步的输出然后用于对输入序列进行分类的预测。 Transformer 解码器中的自我注意操作使用"因果"masking 模式。在生成输出序列的第 i 个entity时，因果 masking 会阻止模型关注 j 输入序列的第 $j > i$ 个entity。这在训练期间使用，因此模型在产生输出时无法"预见未来"。这种 masking 模式的注意力矩阵如图 3 中间所示。

编码器解码器 Transformer 中的解码器用于自回归产生输出序列。也就是说，在每个输出时间步，从模型的预测分布中采样 token，并将样本反馈回模型以生成对下一个输出时间步的预测，依此类推。因此，Transformer 解码器（没有编码器）可以用作语言模型 (LM)，即专门为下一步预测训练的模型 (Liu et al., 2018; Radford et al., 2018; Al -Rfou 等人，2019 年）。这构成了我们考虑的第二个模型结构。该架构的示意图如图 4 中部所示。事实上，NLP 迁移学习的早期工作使用这种具有语言建模目标的架构作为预训练方法（Radford 等人，2018 年）。

语言模型通常用于压缩或序列生成（Graves，2013）。然而，它们也可以简单地通过 concat 启用输入和目标来用于文本到文本框架。例如，考虑英语到德语翻译的情况：如果我们有一个输入句子"That is good"的训练数据点。并以"Das ist gut."为目标，我们将简单地训练模型对 concat 启用的输入序列"translate English to German: That is good. target: Das ist gut."进行下一步预测。如果我们想要获得模型对这个例子的预测，模型将被输入前缀"translate English to German: That is good. target:"并且将被要求自回归生成序列的其余部分。这样，模型可以在给定输入的情况下预测输出序列，满足文本到文本任务的需要。这种方法最近被用来表明语言模型可以在没有监督的情况下学习执行一些文本到文本的任务（Radford 等人，2019 年）。

在 text-to-text 设置中使用语言模型的一个基本且经常被引用的缺点是因果 masking 强制模型对输入序列的 i entity的表示仅取决于 i 之前的entity.要了解为什么这可能是不利的，请考虑文本到文本框架，其中在要求模型进行预测之前为模型提供前缀/上下文（例如，前缀是英语句子，而模型被要求预测德语翻译）。通过完全因果 masking，模型对前缀状态的表示只能依赖于前缀的先前entity。因此，在预测输出的entity时，模型将注意不必要地限制的前缀表示。类似的论点也反对在序列到序列模型中使用单向递归神经网络编码器 (Bahdanau et al., 2015)。

在基于 Transformer 的语言模型中，只需更改 masking 模式即可避免此问题。我们没有使用因果 mask，而是在序列的前缀部分使用完全可见的 masking。这种 masking 模式和由此产生的"前缀 LM"（我们考虑的第三种模型结构）的示意图分别在图 3 和图 4 的最右侧面板中进行了说明。在上面提到的英语到德语的翻译样本中，完全可见的 masking 将应用于前缀"translate English to German: That is good.target:"，并且在训练期间将使用因果 masking 来预测瞄准"Das ist gut"。在文本到文本框架中使用前缀 LM 最初是由 Liu 等人提出的(2018)。最近，董等人(2019) 表明这种架构对各种文本到文本的任务都很有效。该架构类似于编码器-解码器模型，其参数在编码器和解码器之间共享，并且编码器-解码器注意力被输入和目标序列的完全注意力所取代。

我们注意到，当遵循我们的文本到文本框架时，前缀 LM 架构非常类似于用于分类任务的 BERT（Devlin 等人，2018）。要了解原因，请考虑 MNLI 基准测试中的一个样本，其中前提是“我讨厌鸽子”，假设是“我对鸽子的感觉充满了敌意”。正确的标签是“蕴含”。要将此样本输入语言模型，我们会将其转换为序列“mnli 前提：我讨厌鸽子。假设：我对鸽子的感情充满敌意。目标：蕴含”。在这种情况下，完全可见的前缀将对应整个输入序列，直到单词“target:”，这可以看作类似于 BERT 中使用的“分类”token。因此，我们的模型将对整个输入具有完整的可见性，然后将负责通过输出单词“entailment”来进行分类。给定任务前缀（在本例中为“mnli”），模型很容易学会输出有效类标签之一。因此，前缀 LM 和 BERT 架构之间的主要区别在于，分类器只是简单地集成到前缀 LM 中 Transformer 解码器的输出层中。

3.2.2. 比较不同的模型结构

为了通过实验比较这些架构变体，我们希望我们认为的每个模型在某种有意义的方式上是等效的。如果两个模型具有相同数量的参数，或者它们需要大致相同的计算量来处理给定的（输入序列，目标序列）对，我们可能会说这两个模型是等价的。不幸的是，不可能同时根据这两个标准将编码器-解码器模型与语言模型架构（包含单个 Transformer 堆栈）进行比较。要了解原因，首先请注意在编码器中具有 L 层和在解码器中具有 L 层的编码器-解码器模型与具有 $2L$ 层的语言模型具有大致相同数量的参数。然而，相同的 $L + L$ 编码器-解码器模型将具有与仅具有 L 层的语言模型大致相同的计算成本。这是因为语言模型中的 L 层必须应用于输入和输出序列，而编码器仅应用于输入序列，解码器仅应用于输出序列。请注意，这些等价是近似的——由于编码器-解码器的注意力，解码器中有一些额外的参数，并且在序列长度上是二次方的注意力层中也有一些计算成本。然而，在实践中，我们观察到 L 层语言模型与 $L + L$ 层编码器-解码器模型的步数几乎相同，这表明计算成本大致相同。此外，对于我们考虑的模型大小，编码器-解码器注意层中的参数数量约为总参数计数的 10%，因此我们做出简化假设，即 $L + L$ 层编码器-解码器模型具有与 $2L$ 层语言模型相同数量的参数。

为了提供合理的比较方法，我们考虑了编码器-解码器模型的多种配置。我们将 $\text{BERT}_{\text{BASE}}$ 大小的层堆栈中的层数和参数分别称为 L 和 P。我们将使用 M 来指代 $L + L$ 层编码器-解码器模型或 L 层仅解码器模型处理给定输入-目标对所需的 FLOP 数。总的来说，我们将比较：

- 编码器-解码器模型，编码器中有 L 层，解码器中有 L 层。该模型具有 $2P$ 参数和 M FLOPs 的计算成本。
- 等效模型，但参数在编码器和解码器之间共享，导致 P 参数和 M-FLOP 计算成本。
- 编码器-解码器模型，在编码器和解码器中各有 $L/2$ 层，给出 P 参数和 $M/2$-FLOP 成本。
- 具有 L 层和 P 参数的仅解码器语言模型以及 M FLOPs 的计算成本。
- 具有相同架构（因此具有相同数量的参数和计算成本）的仅解码器前缀 LM，但对输入具有完全可见的自注意力。

3.2.3. 目标

作为无监督目标，我们将同时考虑基本语言建模目标和第 3.1.4 节中描述的baseline去噪目标。我们将语言建模目标包括在内，因为它在历史上被用作预训练目标（Dai 和 Le，2015 年；Ramachandran 等人，2016 年；Howard 和 Ruder，2018 年；Radford 等人，2018 年；Peters 等人，2018 年））以及它自然适合我们考虑的语言模型架构。对于在进行预测之前摄取前缀的模型（编码器-解码器模型和前缀 LM），我们从未标记的数据集中采样一段文本，然后选择一个随机点将其拆分为前缀和目标部分。对于标准语言模型，我们训练模型预测从开始到结束的整个跨度。我们的无监督去噪目标是为文本到文本模型设计的；为了适应语言模型的使用，我们 concat 按照第 3.2.1 节中的描述设置输入和目标。

3.2.4. 结果

表 2 显示了我们比较的每个架构所取得的分数。对于所有任务，具有去噪目标的编码器-解码器架构表现最好。该变体具有最高的参数计数 ($2P$)，但与 P 参数解码器模型的计算成本相同。令人惊讶的是，我们发现在编码器和解码器之间共享参数的效果几乎一样。相反，将编码器和解码器堆栈中的层数减半会严重损害性能。并发工作（Lan 等人，2019 年）还发现，跨 Transformer 块共享参数可以成为在不牺牲太多性能的情况下降低总参数数量的有效方法。 XLNet 还与具有去噪目标的共享编码器-解码器方法有一些相似之处（Yang 等人，2019）。我们还注意到共享参数编码器-解码器优于仅解码器前缀 LM，这表明添加显式编码器-解码器注意力是有益的。最后，我们确认了一个广泛持有的概念，即与语言建模目标相比，使用去噪目标总是能带来更好的下游任务性能。

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	Denoising	P	M	81.82	18.61	78.94	68.11	26.43	37.98	27.39
Encoder-decoder	LM	$2P$	M	79.56	18.59	76.02	64.29	26.27	39.17	26.86
Enc-dec, shared	LM	P	M	79.60	18.13	76.35	63.50	26.62	39.17	27.05
Enc-dec, 6 layers	LM	P	$M/2$	78.67	18.26	75.32	64.06	26.13	38.42	26.89
Language model	LM	P	M	73.78	17.54	53.81	56.51	25.23	34.31	25.38
Prefix LM	LM	P	M	79.68	17.84	76.87	64.86	26.28	37.51	26.76

表 2：第 3.2.2 节中描述的不同架构变体的性能。我们使用 P 来指代 12 层基础 Transformer 层堆栈中的参数数量，并使用 M 来指代使用编码器解码器模型处理序列所需的 FLOP。我们使用去噪目标（在第 3.1.4 节中描述）和自回归目标（通常用于训练语言模型）来评估每个架构变体。

Devlin 等人之前已经进行了这一观察(2018)，Voita 等人(2019)，以及 Lample 和 Conneau (2019) 等。我们在下一节中对无监督目标进行了更详细的探索。

3.3. 无监督目标

无监督目标的选择至关重要，因为它提供了模型获得通用知识以应用于下游任务的机制。这导致了各种各样的预训练目标的发展（Dai 和 Le，2015 年；Ramachandran 等人，2016 年；Radford 等人，2018 年；Devlin 等人，2018 年；Yang 等人，2019 年； Liu et al., 2019b; Wang et al., 2019a; Song et al., 2019; Dong et al., 2019; Joshi et al., 2019)。在本节中，我们对无监督目标的空间进行程序探索。在许多情况下，我们不会完全复制现有的目标——一些将被修改以适应我们的文本到文本编码器-解码器框架，在其他情况下，我们将使用结合了多种常见方法的概念的目标。

总的来说，我们所有的目标都从我们未标记的文本数据集中提取了一系列 token ID，这些 ID 对应于 token 化的文本范围。 token 序列被处理以产生（破坏的）输入序列和相应的目标。然后，像往常一样以最大似然训练模型来预测目标序列。我们在表 3 中提供了我们考虑的许多目标的说明性样本。

3.3.1. 不同的高级方法

首先，我们比较了三种受常用目标启发但方法大不相同的技术。首先，我们包括一个基本的“前缀语言建模”目标，如第 3.2.3 节中所用。该技术将一段文本分成两个部分，一个用作编码器的输入，另一个用作解码器预测的目标序列。

Objective	Inputs	Targets
Prefix language modeling	Thank you for inviting	me to your party last week .
BERT-style Devlin et al. (2018)	Thank you `<M>` `<M>` me to your party apple week .	*(original text)*
Deshuffling	party me for your to . last fun you inviting week Thank	*(original text)*
MASS-style Song et al. (2019)	Thank you `<M>` `<M>` me to your party `<M>` week .	*(original text)*
I.i.d. noise, replace spans	Thank you `<X>` me to your party `<Y>` week .	`<X>` for inviting `<Y>` last `<Z>`
I.i.d. noise, drop tokens	Thank you me to your party week .	for inviting last
Random spans	Thank you `<X>` to `<Y>` week .	`<X>` for inviting me `<Y>` your party last `<Z>`

表 3：我们认为应用于输入文本“感谢您上周邀请我参加您的聚会”的一些无监督目标产生的输入和目标样本。请注意，我们所有的目标都处理 token 化的文本。对于这个特定的句子，我们的词汇表将所有单词映射到一个 token。我们将（原始文本）写为目标，表示该模型的任务是重建整个输入文本。⟨M⟩ 表示共享 mask token 和 ⟨X⟩, ⟨Y⟩，⟨Z⟩ 表示分配了唯一 tokens ID 的哨兵 token。 BERT 风格的目标（第二行）包含一个破坏，其中一些 tokens 被随机 token ID 替换；我们通过灰色词苹果来展示这一点。

Objective	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
Prefix language modeling	80.69	18.94	77.99	65.27	**26.86**	39.73	**27.49**
BERT-style (Devlin et al., 2018)	**82.96**	**19.17**	**80.65**	**69.85**	**26.78**	**40.03**	**27.41**
Deshuffling	73.17	18.59	67.61	58.47	26.11	39.30	25.62

表 4：第 3.3.1 节中描述的三个不同的预训练目标的性能。

其次，我们考虑一个受 BERT 中使用的“masked 语言建模”(MLM) 目标启发的目标（Devlin 等人，2018 年）。 MLM 获取一段文本并破坏 tokens 的 15%。破坏的 tokens 中的 90% 被替换为特殊的 mask token 和 10% 被随机 token 替换。由于 BERT 是一个仅包含编码器的模型，因此它在预训练期间的目标是在编码器的输出端重建 masked tokens。在编码器-解码器的情况下，我们简单地使用整个未破坏的序列作为目标。请注意，这与我们的基准目标不同，后者仅使用破坏的 tokens 作为目标；我们在 3.3.2 节中比较了这两种方法。最后，我们还考虑了一个基本的去打乱目标，例如在 (Liu et al., 2019a) 中，它被应用于去噪顺序自动编码器。这种方法采用 tokens 序列，对其进行打乱，然后使用原始的去打乱序列作为目标。我们在表 3 的前三行中提供了这三种方法的输入和目标样本。

这三个目标的性能如表 4 所示。总的来说，我们发现 BERT 风格的目标表现最好，尽管前缀语言建模目标在翻译任务上获得了相似的性能。事实上，BERT 目标的动机是超越基于语言模型的预训练。去打乱目标的性能比前缀语言建模和 BERT 风格的目标都要差得多。

Objective	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
BERT-style (Devlin et al., 2018)	82.96	19.17	**80.65**	69.85	26.78	**40.03**	27.41
MASS-style (Song et al., 2019)	82.32	19.16	80.10	69.28	26.79	**39.89**	27.55
★ Replace corrupted spans	83.28	**19.24**	**80.88**	**71.36**	**26.98**	39.82	**27.65**
Drop corrupted tokens	**84.44**	**19.31**	**80.52**	68.67	**27.07**	39.76	**27.82**

表 5：BERT 式预训练目标变体的比较。在前两个变体中，模型被训练来重建原始未破坏的文本段。在后两者中，模型仅预测破坏的 tokens 序列。

3.3.2. 简化 BERT 目标

基于上一节的结果，我们现在将重点探索对 BERT 式去噪目标的修改。这个目标最初是作为一种预训练技术提出的，用于为分类和跨度预测训练的仅编码器模型。因此，可以对其进行修改，使其在我们的编码器-解码器文本到文本设置中表现更好或更高效。

首先，我们考虑 BERT 风格目标的一个简单变体，其中我们不包括随机 token 交换步骤。生成的目标只是用 tokens mask 替换输入中 token 的 15%，并且训练模型以重建原始未破坏的序列。 Song 等人使用了类似的 masking 目标(2019) 它被称为“MASS”，因此我们将此变体称为“MASS 式”目标。其次，我们有兴趣看看是否有可能避免预测整个未破坏的文本范围，因为这需要在解码器中对长序列进行自我关注。我们考虑两种策略来实现这一点：首先，我们不是用 token mask 替换每个破坏的 token，而是用唯一的 tokens mask 替换每个连续的破坏 token 范围的整体。然后，目标序列成为“破坏”跨度的 concatenation，每个都以 mask token 为前缀，用于在输入中替换它。这是我们在baseline 中使用的预训练目标，如第 3.1.4 节所述。其次，我们还考虑了一种变体，我们简单地从输入序列中

完全删除破坏的 tokens，并让模型按顺序重建删除的 tokens。这些方法的样本显示在表 3 的第五行和第六行中

表 5 显示了原始 BERT 风格目标与这三种替代方案的实证比较。我们发现，在我们的设置中，所有这些变体的表现都相似。唯一的例外是，由于 CoLA 得分显着更高（60.04，与我们的baseline平均值 53.84 相比，参见表 16），完全丢弃破坏的 tokens 会在 GLUE 得分上产生小幅改善。这可能是由于 CoLA 涉及对给定句子在语法和句法上是否可接受进行分类，并且能够确定 tokens 何时缺失与检测可接受性密切相关。然而，在 SuperGLUE 上完全丢弃 tokens 比用哨兵 tokens 替换它们表现更差。不需要预测完整原始序列的两个变体（“替换破坏的跨度”和“删除破坏的跨度”）都具有潜在的吸引力，因为它们使目标序列更短，从而使训练

破坏率	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
10%	**82.82**	19.00	**80.38**	69.55	**26.87**	39.28	**27.44**
⋆15%	**83.28**	19.24	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
25%	**83.00**	**19.54**	**80.96**	70.48	**27.04**	**39.83**	**27.47**
50%	81.27	19.32	79.80	70.33	**27.01**	**39.90**	**27.49**

表 6：独立同分布的性能具有不同破坏率的破坏目标。

快点。展望未来，我们将探索用哨兵 tokens 替换破坏的跨度并仅预测破坏的 tokens（如我们的基准目标）的变体。

3.3.3. 改变破坏率

到目前为止，我们一直在破坏 tokens 的 15%，这是 BERT 中使用的值（Devlin 等人，2018）。同样，由于我们的文本到文本框架与 BERT 的不同，我们有兴趣看看不同的破坏率是否更适合我们。我们在表 6 中比较了 10%、 15%, 25% 和 50% 的破坏率。总体而言，我们发现破坏率对模型性能的影响有限。唯一的例外是我们考虑的最大破坏率 (50%) 导致 GLUE 和 SQuAD 的性能显着下降。使用更大的破坏率也会导致更长的目标，这可能会减慢训练速度。基于这些结果和 BERT 设定的历史先例，我们将使用 15% 的破坏率。

3.3.4. 破坏跨度

我们现在转向通过预测更短的目标来加速训练的目标。到目前为止，我们使用的方法是独立同分布。决定每个输入 token 是否破坏它。当多个连续的 tokens 被破坏时，它们被视为一个“跨度”，并且使用一个唯一的 mask token 来替换整个跨度。用单个 token 替换整个跨度会导致未标记的文本数据被处理成更短的序列。由于我们使用的是独立同分布。破坏策略，大量破坏的 tokens 连续出现的情况并不总是如此。因此，我们可能会通过专门破坏 tokens 的跨度而不是破坏独立同分布中的单个 tokens 来获得额外的加速。方式。 Corrupting spans 以前也被认为是 BERT 的预训练目标，它被发现可以提高性能（Joshi 等人， 2019)。

为了测试这个想法，我们考虑了一个专门破坏连续的、随机间隔的令牌跨度的目标。这个目标可以通过被破坏的令牌的比例和被破坏的跨度的总数来参数化。然后随机选择跨度长度以满足这些指定参数。例如，如果我们正在处理一个包含 500 个标记的序列，并且我们指定 15% 的标记应该破坏并且应该有 25 个总跨度，那么破坏的标记总数将为 $500\times 0.15=75 $，平均跨度长度为 $75/25 = 3$ 。请注意，给定原始序列长度和破坏率，我们可以通过平均跨度长度或跨度总数来等效地参数化该目标。

跨度	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
⋆ baseline (i.i.d.)	**83.28**	19.24	80.88	71.36	**26.98**	**39.82**	**27.65**
2	**83.54**	19.39	**82.09**	**72.20**	**26.76**	**39.99**	**27.63**
3	**83.49**	**19.62**	**81.84**	**72.53**	**26.86**	39.65	**27.62**
5	**83.40**	19.24	**82.05**	**72.23**	**26.88**	39.40	**27.53**
10	82.85	19.33	**81.84**	70.44	**26.79**	39.49	**27.69**

表 7：不同平均跨度长度的跨度破坏目标的性能（受 Joshi 等人 (2019) 的启发）。在所有情况下，我们都会破坏原始文本序列的 15%。

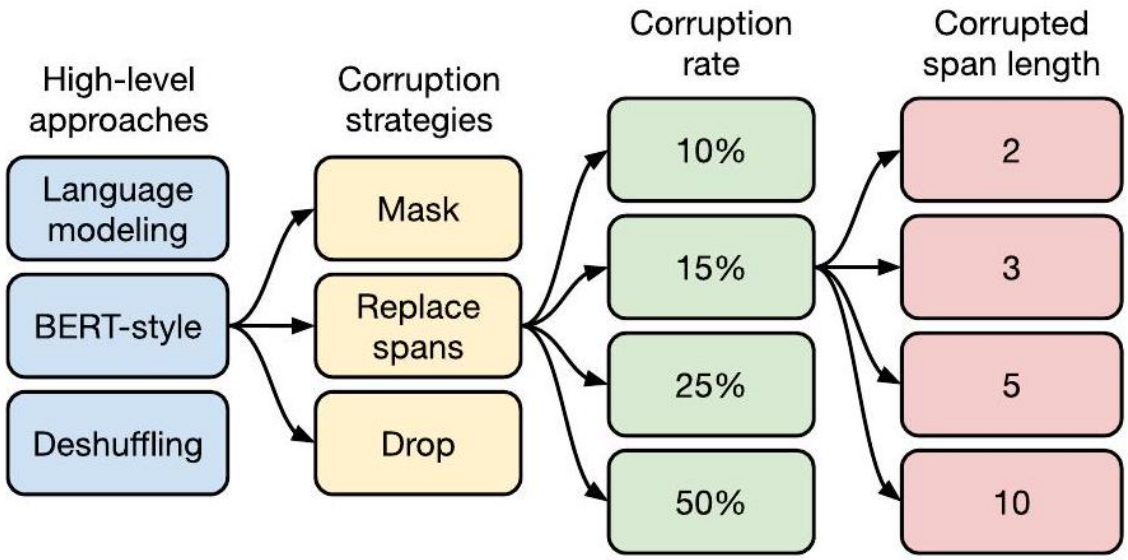

图 5：我们探索无监督目标的流程图。我们首先考虑第 3.3.1 节中的几种不同方法，发现 BERT 风格的去噪目标表现最好。然后，我们考虑各种方法来简化 BERT 目标，以便它在第 3.3.2 节中产生更短的目标序列。鉴于用哨兵 tokens 替换掉落的跨度表现良好并导致短目标序列，在第 3.3.3 节中，我们试验不同的破坏率。最后，我们在第 3.3.4 节中评估了一个故意破坏 tokens 连续跨度的目标。

我们在表 7 中将跨度破坏目标与 i.i.d-破坏目标进行了比较。我们在所有情况下都使用 15% 的破坏率，并使用 2,3、5 和 10 的平均跨度长度进行比较。同样，我们发现一个有限的这些目标之间的差异，尽管在某些情况下平均跨度长度为 10 的版本略低于其他值。我们还特别发现，使用 3 的平均跨度长度略微（但显着）优于独立同分布。大多数非翻译基准的目标。幸运的是，与 i.i.d. 相比，span-corruption 目标在训练期间也提供了一些加速。噪声方法，因为跨度破坏平均会产生较短的序列。

3.3.5. 讨论

图 5 显示了我们在探索无监督目标期间所做选择的流程图。总的来说，我们观察到的最显着的性能差异是去噪目标优于语言建模和预训练去混洗。在我们探索的去噪目标的许多变体中，我们没有观察到显着差异。然而，不同的目标（或目标的参数化）可能导致不同的序列长度，从而导致不同的训练速度。这意味着我们在这里考虑的去噪目标之间的选择应该主要根据它们的计算成本来完成。我们的结果还表明，对与我们在此考虑的目标类似的目标进行额外探索可能不会为我们考虑的任务和模型带来显着收益。相反，探索利用未标记数据的完全不同的方法可能是偶然的。

3.4. 预训练数据集

与无监督目标一样，预训练数据集本身是迁移学习 pipeline 的重要组成部分。然而，与目标和基准不同的是，新的预训练数据集本身通常不会被视为重要贡献，并且通常不会与预训练模型和代码一起发布。相反，它们通常是在介绍新方法或模型的过程中引入的。因此，不同预训练数据集的比较相对较少，也缺乏用于预训练的"标准"数据集。最近一些值得注意的例外（Baevski 等人，2019 年；Liu 等人，2019c；Yang 等人，2019 年）将对新的大型（通常是 Common Crawl 来源的）数据集的预训练与使用较小的现有数据集进行了比较集（通常是维基百科）。为了更深入地探讨预训练数据集对性能的影响，在本节中，我们比较了 C4 数据集的变体和其他潜在的预训练数据源。我们发布了所有我们认为是 TensorFlow 数据集一部分的 C4 数据集变体。 [11]

3.4.1. 未标记的数据集

在创建 C4 时，我们开发了各种启发式方法来过滤从 Common Crawl 中提取的网络文本（有关描述，请参见第 2.2 节）。除了将其与其他过滤方法和常见的预训练数据集进行比较之外，我们还有兴趣衡量这种过滤是否会提高下游任务的性能。为此，我们在以下数据集上进行预训练后比较了baseline模型的性能：

C4 作为baseline，我们首先考虑对我们提出的未标记数据集进行预训练，如第 2.2 节所述

未过滤的 C4 为了衡量我们在创建 C4 时使用的启发式过滤的效果（重复数据删除、删除不良词、仅保留句子等），我们还生成了放弃此过滤的 C4 的替代版本。请注意，我们仍然使用 langdetect 来提取英文文本。因此，我们的"未过滤"变体仍然包含一些过滤，因为 langdetect 有时会将低概率分配给非自然英语文本。

RealNews-like 最近的工作使用了从新闻网站中提取的文本数据（Zellers 等人，2019 年；Baevski 等人，2019 年）。为了与这种方法进行比较，我们通过额外过滤 C4 以仅包含来自"RealNews"数据集中使用的一个域的内容来生成另一个未标记的数据集（Zellers 等人，2019）。请注意，为了便于

11. https://www.tensorflow.org/datasets/catalog/c4

数据集	大小	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	E
†44	745 GB	83.28	**19.24**	80.88	71.36	**26.98**	**39.82**	**2**
C4，未过滤	6.1 TB	81.46	19.14	78.78	68.04	26.55	39.34	2
RealNews-like	35 GB	**83.83**	**19.23**	80.39	72.38	**26.75**	**39.90**	**2**
WebText-like	17 GB	**84.03**	**19.31**	**81.42**	71.40	**26.80**	**39.74**	**2**
维基百科	16 GB	81.85	**19.31**	81.29	68.01	**26.94**	39.69	**2**
维基百科 + TBC	20 GB	83.65	**19.28**	**82.08**	**73.24**	**26.77**	39.63	**2**

表 8：不同数据集预训练的性能。前四个变体基于我们新的 C4 数据集。

比较，我们保留了C4中使用的启发式过滤方法；唯一的区别是我们表面上省略了任何非新闻内容。

与 WebText-like 类似，WebText 数据集（Radford 等人，2019）仅使用提交给内容聚合网站 Reddit 且"分数"至少为 3 的网页内容。提交给 Reddit 的网页的分数是根据支持（赞成）或反对（反对）该网页的用户比例计算的。使用 Reddit 分数作为质量信号背后的想法是，该网站的用户只会对高质量的

文本内容进行投票。为了生成可比较的数据集，我们首先尝试从 C4 中删除并非源自 OpenWebText 准备的列表中出现的 URL 的所有内容。 [12] 然而，这导致内容相对较少——只有大约 2 GB——因为大多数页面从未出现在 Reddit 上。回想一下，C4 是基于单月的 Common Crawl 数据创建的。为了避免使用过小的数据集，我们因此从 2018 年 8 月到 2019 年 7 月从 Common Crawl 下载了 12 个月的数据，对 C4 应用了我们的启发式过滤，然后应用了 Reddit 过滤器。这产生了一个 17 GB 的类似 WebText 的数据集，其大小与原始的 40GB WebText 数据集相当（Radford 等人，2019）。

Wikipedia Wikipedia 网站由数百万协作编写的百科全书文章组成。网站上的内容遵循严格的质量准则，因此已被用作干净自然文本的可靠来源。我们使用来自 TensorFlow Datasets 的英文维基百科文本数据。 [13] 省略了文章中的任何标记或参考部分。

Wikipedia + Toronto Books Corpus 使用来自维基百科的预训练数据的一个缺点是它只代表自然文本（百科全书文章）的一个可能域。为了缓解这种情况，BERT（Devlin 等人，2018 年）将来自维基百科的数据与多伦多图书语料库（TBC）（Zhu 等人，2015 年）结合起来。 TBC 包含从电子书中提取的文本，代表自然语言的不同领域。 BERT 的火爆使得 Wikipedia + TBC 的组合被用于后续的很多工作中。

12. https://github.com/jcpeterson/openwebtext
13. https://www.tensorflow.org/datasets/catalog/wikipedia

表 8 显示了对每个数据集进行预训练后取得的结果。第一个明显的收获是从 C4 中移除启发式过滤会统一降级性能并使未过滤的变体在每项任务中表现最差。除此之外，我们发现在某些情况下，具有更多约束域的预训练数据集优于多样化的 C4 数据集。例如，使用维基百科 + TBC 语料库产生的 SuperGLUE 得分为 73.24，超过了我们的baseline得分（使用 C4）71.36。这几乎完全归因于 MultiRC 的精确匹配分数从 25.78（baseline，C4）提高到 50.93（维基百科 + TBC）（见表 16）。MultiRC是一个阅读理解数据集，其最大的数据来源来自于小说类书籍，这正是TBC所涵盖的领域。同样，使用类似 RealNews 的数据集进行预训练使 ReCoRD 的精确匹配分数从 68.16 提高到 73.72，ReCoRD 是衡量新闻文章阅读理解的数据集。作为最后一个例子，使用来自维基百科的数据在 SQuAD 上产生了显着（但不那么显着）的收益，这是一个问答数据集，其段落来自维基百科。在先前的工作中已经进行了类似的观察，例如Beltagy 等人(2019) 发现，对研究论文文本的预训练 BERT 提高了其在科学任务上的表现。这些发现背后的主要教训是，对域内未标记数据进行预训练可以提高下游任务的性能。如果我们的目标是预训练一个可以快速适应来自任意领域的语言任务的模型，这并不令人惊讶，但也不令人满意。刘等人(2019c) 还观察到，对更多样化的数据集进行预训练可以改善下游任务。这一观察也激发了自然语言处理领域适应研究的平行线；有关该领域的调查，请参见例如罗德 (2019)；李 (2012)。

仅在单个域上进行预训练的一个缺点是生成的数据集通常要小得多。类似地，虽然在我们的baseline设置中，类似 WebText 的变体的性能与 C4 数据集一样好或更好，但基于 Reddit 的过滤产生的数据集比 C4 小 $40\times$，尽管它基于 $12\times$ 更多数据来自 Common Crawl。但是请注意，在我们的baseline设置中，我们仅在 $2^{35} \approx 34\,\text{B}$ tokens 上进行预训练，这仅比我们考虑的最小预训练数据集大 8 倍左右。我们将在下一节中调查在什么时候使用较小的预训练数据集会造成问题。

3.4.2. 预训练数据集大小

我们用来创建 C4 的 pipeline 旨在能够创建非常大的预训练数据集。访问如此多的数据使我们能够在不重复样本的情况下预训练我们的模型。目前尚不清楚在预训练期间重复样本是否会对下游性能有帮助或有害，因为我们的预训练目标本身是随机的，可以帮助防止模型多次看到相同的精确数据。

为了测试有限的未标记数据集大小的影响，我们在 C4 的人工截断版本上预训练了我们的baseline模型。回想一下，我们在 $2^{35} \approx 34\,\text{B}$ tokens（ C4 总大小的一小部分）上预训练了我们的baseline模型。我们考虑对由 C4 和 $2^{29}, 2^{27}, 2^{25}$ tokens 组成的 2^{23} 的截断变体进行训练。这些大小对应于在预训练过程中分别重复数据集 $64, 256, 1,024$ 和 $4{,}096$ 次。

tokens	重复次数	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr
* 完整数据集	0	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**
2^{29}	64	**82.87**	**19.19**	**80.97**	**72.03**	**26.83**	**39.74**
2^{27}	256	82.62	**19.20**	79.78	69.97	**27.02**	**39.71**
2^{25}	1,024	79.55	18.57	76.27	64.76	26.38	39.56
2^{23}	4,096	76.34	18.33	70.92	59.29	26.37	38.84

表 9：测量预训练期间重复数据的效果。在这些实验中，我们仅使用 C4 中的第一个 N tokens（第一列中显示了不同的 N 值），但仍然在 2^{35} tokens 上进行了预训练。这导致数据集在预训练过程中被重复（第二列显示每个实验的重复次数），这可能导致记忆（见图 6）。

表 9 显示了由此产生的下游性能。正如预期的那样，随着数据集大小的缩小，性能会下降。我们怀疑这可能是因为模型开始记忆预训练数据集。为了衡量这是否属实，我们在图 6 中绘制了每个数据集大小的训练损失。实际上，随着预训练数据集规模的缩小，模型的训练损失显着减少，这表明可能需要记忆。巴耶夫斯基等人(2019) 同样观察到截断预训练数据集的大小会降低下游任务的性能。

我们注意到，当预训练数据集仅重复 64 次时，这些效果是有限的。这表明一定程度的重复预训练数据可能无害。然而，鉴于额外的预训练可能是有益的（正如我们将在第 3.6 节中展示的那样）并且获得额外的未标记数据既便宜又容易，我们建议尽可能使用大型预训练数据集。我们还注意到，这种效果对于较大的模型尺寸可能更为明显，即较大的模型可能更容易过度拟合较小的预训练数据集。

3.5. 训练策略

到目前为止，我们已经考虑了模型的所有参数都在无监督任务上进行预训练的设置，然后再对单个监督任务进行 fine-tuned。虽然这种方法很简单，但已经提出了各种用于在下游/监督任务上训练模型的替代方法。在本节中，除了在多个任务上同时训练模型的方法之外，我们还比较了 fine-tuning 模型的不同方案。

3.5.1. 微调方法

有人认为 fine-tuning 模型的所有参数都可能导致次优结果，尤其是在低资源任务上（Peters 等人，2019 年）。文本分类任务迁移学习的早期结果提倡 fine-tuning 仅使用由固定预训练模型生成的句子 embeddings 提供的小型分类器的参数（Subramanian 等人，2018 年；Kiros 等人，2015 年；Logeswaran 和 Lee，2018 年；Hill 等人，2016 年；Conneau

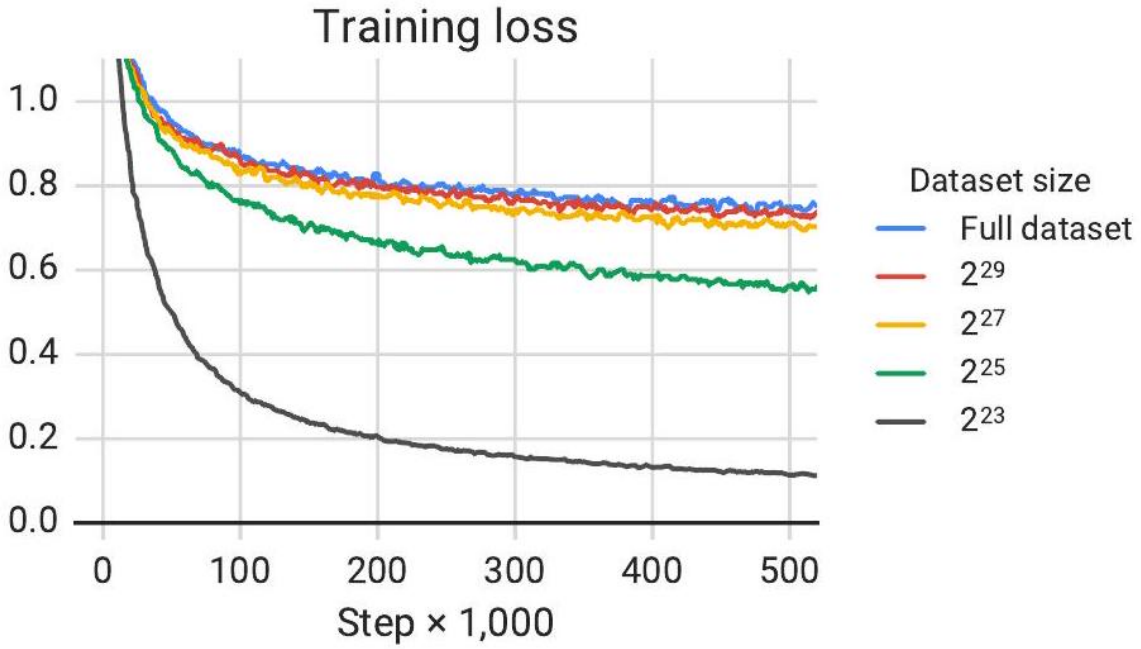

图 6：我们原始 C4 数据集以及 4 个人工截断版本的预训练损失。列出的大小是指每个数据集中 tokens 的数量。所考虑的四种大小对应于在预训练过程中将数据集重复 64 到 4,096 次。使用较小的数据集大小会导致较小的训练损失值，这可能表明对未标记的数据集有一些记忆。

等人，2017 年）。这种方法不太适用于我们的编码器-解码器模型，因为必须训练整个解码器以输出给定任务的目标序列。相反，我们专注于两种替代 fine-tuning 方法，它们仅更新编码器-解码器模型的参数子集。

第一个是“适配器层”（Houlsby 等人，2019 年；Bapna 等人，2019 年），其动机是在 fine-tuning 时保持大部分原始模型固定。适配器层是额外的 dense-ReLU-dense 块，在 Transformer 的每个块中的每个预先存在的前馈网络之后添加。这些新的前馈网络的设计使其输出维度与输入相匹配。这允许将它们插入网络而无需对结构或参数进行额外更改。微调时，仅更新适配层和层归一化参数。这种方法的主要超参数是前馈网络的内部维度 d，它改变了添加到模型中的新参数的数量。我们对 d 的各种值进行了实验。

我们考虑的第二种替代 fine-tuning 方法是“逐渐解冻”（Howard 和 Ruder，2018 年）。在逐渐解冻中，越来越多的模型参数会随着时间的推移进行微调。渐进解冻最初应用于由单层堆栈组成的语言模型架构。在这个设置中，在 fine-tuning 开始时只更新最后一层的参数，然后在训练一定数量的更新后，倒数第二层的参数也包括在内，依此类推直到整个网络参数正在 fine-tuned。为了使这种方法适用于我们的编码器-解码器模型，我们逐渐并行解冻编码器和解码器中的层，在这两种情况下都从顶部开始。由于我们的输入 embedding 矩阵和输出分类矩阵的参数是共享的，因此我们在整个 fine-tuning 中更新它们。回想一下，我们的baseline模型在编码器和解码器中各包含 12 层，并且是 fine-tuned

微调方法	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
所有参数	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
适配器层，$d = 32$	80.52	15.08	79.32	60.40	13.84	17.88	15.54
适配器层，$d = 128$	81.51	16.62	79.47	63.03	19.83	27.50	22.63
适配器层，$d = 512$	81.54	17.78	79.18	64.30	23.45	33.98	25.81
适配器层，$d = 2048$	81.51	16.62	79.47	63.03	19.83	27.50	22.63
逐步解冻	82.50	18.95	79.17	**70.79**	26.71	39.02	26.93

表 10：仅更新模型参数子集的不同备选 fine-tuning 方法的比较。对于适配器层， d 指的是适配器的内部维度。

2^{18} 步骤。因此，我们将 fine-tuning 过程细分为 12 集 $2^{18}/12$ 步骤，并在第 $12 - n$ 集中从层 n 训练到第 12 层。我们注意到 Howard 和 Ruder (2018) 建议在每次 fine-tuning 训练后 epoch 增加一层。然而，由于我们的监督数据集在大小上变化很大，并且由于我们的一些下游任务实际上是许多任务（GLUE 和 SuperGLUE）的混合，我们改为采用更简单的策略，即在每 $2^{18}/12$ 步骤之后添加一个额外的层 fine-tuning .

表 10 显示了这些 fine-tuning 方法的性能比较。对于适配器层，我们使用 32,128、512、2048 的内部维度 d 报告性能。根据过去的结果（Houlsby 等人，2019 年）；Bapna et al., 2019）我们发现像 SQuAD 这样的低资源任务在较小的 d 值下工作良好，而高资源任务需要大维度才能实现合理的性能。这表明，只要维度适当地缩放到任务大小，适配器层可能是 fine-tuning 的一种有前途的技术，参数更少。请注意，在我们的例子中，我们通过 concat 启用它们的组成数据集，将 GLUE 和 SuperGLUE 每个都视为一个“任务”，因此尽管它们包含一些低资源数据集，但组合数据集足够大，因此需要一个大的值 d。我们发现逐渐解冻会导致所有任务的性能略有下降，尽管它确实在 fine-tuning 期间提供了一些加速。通过更仔细地调整解冻 schedule 可以获得更好的结果。

3.5.2. 多任务学习

到目前为止，我们一直在针对单个无监督学习任务对我们的模型进行预训练，然后再 fine-tuning 单独针对每个下游任务进行训练。另一种方法称为“多任务学习”（Ruder，2017 年；Caruana，1997 年），是一次在多个任务上训练模型。这种方法的目标通常是训练一个可以同时执行许多任务的单一模型，即模型及其大部分参数在所有任务中共享。我们稍微放宽了这个目标，而是研究一次对多个任务进行训练的方法，以便最终生成在每个单独任务上表现良好的单独参数设置。例如，我们可能会在多个任务上训练一个模型，但在报告性能时，我们可以为每个任务选择不同的 checkpoint。与我们目前考虑的 pre-train-then-fine-tune 方法相比，这放松了多任务学习框架并使其处于更平稳的位置。我们还注意到，在我们统一的文本到文本框架中，“多任务学习”简单地对应于将数据集混合在一起。因此，通过将无监督任务视为混合在一起的任务之一，我们仍然可以在使用多任务学习时对未标记数据进行训练。相比之下，多任务学习在 NLP 中的大多数应用都添加了任务特定的分类网络或为每个任务使用不同的损失函数（Liu 等人，2019b）。

正如 Arivazhagan 等人指出的那样(2019)，多任务学习中一个极其重要的因素是每个任务应该训练模型的数据量。我们的目标是不要训练不足或过度训练模型——也就是说，我们希望模型从给定任务中看到足够多的数据，以便它可以很好地执行任务，但又不会看到太多数据，以至于它会记住训练集.如何准确地设置来自每个任务的数据比例取决于各种因素，包括数据集大小、学习任务的“难度”（即模型在能够有效执行任务之前必须看到多少数据）、正则化等。另一个问题是“任务干扰”或“负迁移”的可

能性，其中在一项任务上取得良好表现可能会阻碍另一项任务的表现。鉴于这些担忧，我们首先探索各种策略来设置来自每个任务的数据比例。 Wang 等人进行了类似的探索(2019a)。

样本-比例混合模型对给定任务的过度拟合速度的一个主要因素是任务的数据集大小。因此，设置混合比例的一种自然方法是根据每个任务的数据集的大小按比例进行采样。这相当于 concat 为所有任务启用数据集并从组合数据集中随机抽样样本。但是请注意，我们包括了我们的无监督去噪任务，它使用的数据集比其他任务的数据集大几个数量级。因此，如果我们简单地按每个数据集的大小比例进行采样，模型看到的绝大多数数据将没有标签，并且它将在所有监督任务上训练不足。即使没有无监督任务，一些任务（例如 WMT 英语到法语）也非常大，它们同样会挤出大部分 batches。为了解决这个问题，我们在计算比例之前对数据集大小设置了一个人为的"限制"。具体来说，如果每个 N 任务的数据集中的样本数量是 $e_n, n \in \{1, \ldots, N\}$，那么我们将在训练期间从第 m 个任务中抽取样本的概率设置为 $r_m = \min(e_m, K)/\sum \min(e_n, K)$，其中 K 是人工数据设置大小限制。

温标混合 减轻数据集大小之间巨大差异的另一种方法是调整混合速率的"温度"。多语言 BERT 使用这种方法来确保模型在低资源语言上得到充分训练。 [14] 为了用温度 T 实现温度缩放，我们将每个任务的混合率 r_m 提高到 $1/T$ 的幂并重新归一化速率，使它们总和为 1 。当 $T = 1$时，这种方法相当于样本比例混合，并且随着 T的增加，比例变得更接近均等混合。我们保留数据集大小限制 K（应用于在温度缩放之前获得 r_m），但将其设置为较大的 $K = 2^{21}$ 值。我们使用较大的 K 值，因为升高温度会降低最大数据集的混合率。

14. https://github.com/google-research/bert/blob/master/multilingual.md

混合策略	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
* baseline（训练前/fine-tune）	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
等于	76.13	19.02	76.51	63.37	23.89	34.31	26.78
样本比例，$K = 2^{16}$	80.45	19.04	77.25	69.95	24.35	34.99	27.10
样本比例，$K = 2^{17}$	81.56	19.12	77.00	67.91	24.36	35.00	27.25
样本比例，$K = 2^{18}$	81.67	19.07	78.17	67.94	24.57	35.19	27.39
样本比例，$K = 2^{19}$	81.42	**19.24**	79.78	67.30	25.21	36.30	**27.76**
样本比例，$K = 2^{20}$	80.80	**19.24**	**80.36**	67.38	25.66	36.93	**27.68**
样本比例，$K = 2^{21}$	79.83	18.79	79.50	65.10	25.82	37.22	27.13
温标，$T = 2$	81.90	**19.28**	79.42	69.92	25.42	36.72	27.20
温标，$T = 4$	80.56	**19.22**	77.99	69.54	25.04	35.82	27.45
温标，$T = 8$	77.21	19.10	77.14	66.07	24.55	35.35	27.17

表 11：使用不同混合策略的多任务训练比较。 Examplesproportional mixing是指根据每个数据集的总大小，从每个数据集中抽取样本，对最大数据集大小有一个人为的限制 (K)。温度标度混合通过温度 T 重新标度采样率。对于温标混合，我们使用 $K = 2^{21}$ 的人工数据集大小限制。

均等混合在这种情况下，我们以相等的概率从每个任务中抽取样本。具体来说，每个 batch 中的每个样本都是从我们训练的数据集中随机均匀采样的。这很可能是一个次优策略，因为该模型将很快在低资源任务上过拟合，而在高资源任务上欠拟合。我们主要将其作为参考点，以了解当比例设置不理想时可能出现的问题。

为了将这些混合策略与我们的baseline pre-train-thenfine-tune 结果进行平等比较，我们训练多任务模型的总步数相同： $2^{19} + 2^{18} = 786{,}432$。结果见表11

总的来说，我们发现在大多数任务上，多任务训练的表现不如 fine-tuning 之后的预训练。特别是"均等"混合策略会导致性能急剧下降，这可能是因为低资源任务过度拟合，高资源任务没有看到足够的数据，或者模型没有看到足够多的未标记数据来学习通用语言能力。对于样本比例混合，我们发现对于大多数任务， K 有一个"最佳点"，模型在该位置获得最佳性能，而较大或较小的 K 值往往会导致较差的性能。例外（对于我们考虑的 K 值的范围）是 WMT 英语到法语的翻译，这是一项高资源任务，它总是受益于更高的混合比例。最后，我们注意到温标混合还提供了一种从大多数任务中获得合理性能的方法， $T = 2$ 在大多数情况下表现最佳。先前已经观察到，多任务模型优于针对每个单独任务训练的单独模型的发现，例如由 Arivazhagan 等人撰写(2019) 和 McCann 等人(2018)，尽管已经证明多任务设置可以在非常相似的任务中带来好处 Liu 等人(2019b)；拉特纳等人(2018)。在下一节中，我们将探索缩小多任务训练与预训练-然后-fine-tune 方法之间差距的方法。

训练策略	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr
（无监督预训练 + fine-tuning	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	39.82
多任务训练	81.42	**19.24**	79.78	67.30	25.21	36.30
多任务预训练 + fine-tuning	**83.11**	**19.12**	**80.26**	**71.03**	**27.08**	39.80
留一法多任务训练	81.98	19.05	79.97	**71.68**	**26.93**	39.79
有监督的多任务预训练	79.93	18.96	77.38	65.36	26.81	**40.13**

表 12：无监督预训练、多任务学习和各种形式的多任务预训练的比较。

3.5.3. 将多任务学习与微调相结合

回想一下，我们正在研究多任务学习的轻松版本，我们在混合任务上训练单个模型，但允许使用模型的不同参数设置 (checkpoints) 来评估性能。我们可以通过考虑以下情况来扩展这种方法：模型一次针对所有任务进行预训练，然后在单个监督任务上进行 fine-tuned。这是"MT-DNN" (Liu et al., 2015, 2019b) 使用的方法，引入时在 GLUE 和其他基准测试中实现了最先进的性能。我们考虑这种方法的三种变体：首先，我们简单地在样本比例混合上预训练模型，在每个单独的下游任务上，在 fine-tuning 之前，人工数据集大小限制为 $K = 2^{19}$。这有助于我们衡量在预训练期间将监督任务与无监督目标一起包括在内是否会给模型带来一些有益的早期接触下游任务。我们可能还希望混合多种监督来源可以帮助预训练模型在适应单个任务之前获得一组更通用的"技能"（笼统地说）。为了直接衡量这一点，我们考虑了第二种变体，我们在相同的样本比例混合（使用 $K = 2^{19}$ ）上预训练模型，除了我们从这个预训练混合中省略了一个下游任务。然后，我们 fine-tune 在预训练期间遗漏的任务模型。我们对我们考虑的每个下游任务重复此操作。我们称这种方法为"留一法"多任务训练。这模拟了真实世界的设置，其中预训练模型在预训练期间未见过的任务上 fine-tuned。请注意，多任务预训练提供了多种监督任务的混合。由于其他领域（例如计算机视觉（Oquab et al., 2014; Jia et al., 2014; Huh et al., 2016; Yosinski et al., 2014））使用监督数据集进行预训练，我们感兴趣看看从多任务预训练混合中省略无监督任务是否仍然产生良好的结果。因此，对于我们的第三个变体，我们使用 $K = 2^{19}$ 考虑的所有监督任务的样本比例混合进行预训练。在所有这些变体中，我们遵循我们的标准程序，在 2^{19} 步骤的 fine-tuning 之前进行 2^{18} 步骤的预训练。

我们在表 12 中比较了这些方法的结果。为了比较，我们还包括了我们的baseline（预训练然后 fine-tune）和标准多任务学习（没有 fine-tuning）的结果 $K = 2^{19}$。我们发现 fine-tuning 经过多任务预训练后的性能与我们的baseline相当。这表明在多任务学习之后使用 fine-tuning 可以帮助减轻第 3.5.2 节中描述的不同混合率之间的一些权衡。有趣的是，“留一法”训练的表现只是稍微差一点，这表明经过各种任务训练的模型仍然可以适应新任务（即多任务预训练可能不会导致显着的任务干扰).最后，有监督的多任务预训练在除了翻译任务之外的所有情况下都表现得更差。这可能表明翻译任务从（英语）预训练中获益较少，而无监督预训练是其他任务的重要因素。

3.6. 缩放

机器学习研究的“惨痛教训”认为，可以利用额外计算的通用方法最终会战胜依赖人类专业知识的方法（Sutton，2019 年；Hestness 等人，2017 年；Shazeer 等人，2017 年；Jozefowicz 等人） ., 2016 年；Mahajan 等人，2018 年；Shazeer 等人，2018 年、2017 年；Huang 等人，2018b；Keskar 等人，2019a)。最近的结果表明，这可能适用于 NLP 中的迁移学习（Liu et al., 2019c; Radford et al., 2019; Yang et al., 2019; Lan et al., 2019），即它已被反复证明与更精心设计的方法相比，扩大规模可以提高性能。但是，有多种可能的扩展方法，包括使用更大的模型、为更多步骤训练模型以及集成。在本节中，我们通过解决以下前提来比较这些不同的方法：“您刚刚获得 4× 更多计算。您应该如何使用它？”

我们从我们的baseline模型开始，该模型具有 220M 参数，并且分别针对 2^{19} 和 2^{18} 步骤进行了预训练和 fine-tuned。编码器和解码器的大小都与“BERT BASE”相似。为了尝试增大模型规模，我们遵循了“BERTLARGE”（Devlin等人，2018）的指南，使用了$\text{dff} = 4096$、$\text{dmodel} = 1024$、$\text{dkv} = 64$和 16头的注意力机制。然后，我们在编码器和解码器中生成两个分别具有 16 层和 32 层的变体，生成具有与原始模型一样多参数的 2× 和 4× 模型。这两个变体也具有大致 2× 和 4× 的计算成本。使用我们的baseline和这两个更大的模型，我们考虑三种使用 4× 尽可能多的计算的方法：训练 4× 尽可能多的步骤，使用 2× 更大的模型训练 2× 尽可能多的步骤，以及训练 4× 训练步骤“baseline”数量的更大模型。当我们增加训练步骤时，为了简单起见，我们同时缩放预训练和 fine-tune 步骤。请注意，当增加预训练步骤的数量时，我们有效地包含了更多的预训练数据，因为 C4 太大以至于即使在训练 2^{23} 步时我们也无法完成一次数据传递。

模型查看 4× 尽可能多的数据的另一种方法是将 batch 大小增加 4 倍。由于更有效的并行化，这可能会导致更快的训练。然而，使用 4× 较大的 batch 大小进行训练可能会产生与 4× 训练相同步数的不同结果（Shallue 等人，2018）。我们包括一个额外的实验，我们用 4× 更大的 batch 大小训练我们的baseline模型来比较这两种情况。

在我们考虑的许多基准测试中，通常的做法是通过使用 ensemble 模型进行训练和评估来获得额外的性能。这提供了一种使用附加计算的正交方式。为了将其他缩放方法与集成进行比较，我们还测量了 4 个单独预训练和 ensembled 模型的 fine-tune 的性能。在将它们输入输出 softmax 非线性以获得聚合预测之前，我们对 ensemble 中的 logits 进行平均。代替预训练 4

缩放策略	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
⋆ baseline	83.28	19.24	80.88	71.36	26.98	39.82	27.65
1× 尺寸，4× 训练步数	85.33	19.33	82.45	74.72	27.08	40.66	27.93
1× 尺码，4× batch 尺码	84.60	19.42	82.52	74.64	27.07	40.60	27.84
2× 大小，2× 训练步数	**86.18**	19.66	**84.18**	77.18	27.52	**41.03**	28.19
4× 大小，1× 训练步数	**85.91**	19.73	**83.86**	**78.04**	27.47	40.71	28.10
4× ensembled	84.77	**20.10**	83.09	71.74	**28.05**	40.53	**28.57**
4× ensembled，仅 fine-tune	84.05	19.57	82.36	71.55	27.55	40.22	28.09

表 13：扩大我们的baseline模型的不同方法的比较。除了集成 fine-tuned 模型之外的所有方法都使用 4× 计算作为baseline。 “大小”是指模型中的参数数量，“训练时间”是指用于预训练和 fine-tuning 的步数。

单独的模型，一个更便宜的替代方案是采用单个预训练模型并生成 4 个单独的 fine-tuned 版本。虽然这不会使用我们的整个 4× 计算预算，但我们也包括此方法以查看它是否产生与其他缩放方法相比具有竞争力的性能。

表 13 显示了应用这些不同缩放方法后实现的性能。不出所料，增加训练时间和/或模型大小会持续改善baseline。在训练 4× 步数或使用 4× 更大的 batch 大小之间没有明显的赢家，尽管两者都是有益的。一般来说，与仅增加训练时间或 batch 大小相比，增加模型大小会导致性能的额外提升。我们没有观察到为 2× 训练一个 2× 更大的模型和在我们研究的任何任务上训练一个 4× 更大的模型之间有很大的区别。这表明增加训练时间和增加模型大小可以作为提高性能的补充手段。我们的结果还表明，集成提供了一种通过规模提高性能的正交且有效的方法。在某些任务（CNN/DM、WMT 英语到德语和 WMT 英语到罗马尼亚语）中，集成 4 个完全独立训练的模型明显优于所有其他缩放方法。一起预训练但 fine-tuned 单独训练的集成模型也比baseline有显着的性能提升，这表明提高性能的成本更低的方法。唯一的例外是 SuperGLUE，其中两种集成方法都没有显着改善baseline。

我们注意到，不同的缩放方法具有与其性能分开的不同权衡。例如，使用更大的模型会使下游 fine-tuning 和推理更加昂贵。相比之下，如果将小型模型应用于许多下游任务，则可以有效地分摊更长时间预训练小型模型的成本。另外，我们注意到集成 N 单独模型的成本与使用具有 $N\times$ 更高计算成本的模型的成本相似。因此，在缩放方法之间进行选择时，对模型最终使用的一些考虑很重要。

3.7. 将它们放在一起

我们现在利用我们系统研究的见解来确定我们可以将流行的 NLP 基准的性能提高到什么程度。我们也有兴趣通过在大量数据上训练更大的模型来探索 NLP 迁移学习的当前限制。我们从我们的baseline训练方法开始，并进行以下更改：

目标 我们换掉 i.i.d.我们baseline中的去噪目标用于第 3.3.4 节中描述的跨度破坏目标，该目标大致受到 SpanBERT 的启发（Joshi 等人，2019）。具体来说，我们使用 3 的平均跨度长度和原始序列的破

坏 15%。我们发现该目标产生了略微更好的性能（表 7），同时由于目标序列长度较短，计算效率略高。

更长时间的训练 我们的baseline模型使用相对较少的预训练 (1/4 与 BERT（Devlin 等人，2018）一样多，是 XLNet（Yang 等人，2019）的 1/16，是 XLNet（Yang 等人，2019）的 1/64就像 RoBERTa（Liu 等人，2019c）等）。幸运的是，C4 足够大，我们可以在不重复数据的情况下训练更长的时间（这可能是有害的，如 3.4.2 节所示）。我们在第 3.6 节中发现额外的预训练确实有帮助，并且增加 batch 大小和增加训练步骤的数量都可以带来这种好处。因此，我们在长度为 512 的 2^{11} 序列的 batch 大小上预训练我们的模型 100 万步，对应于总共约 1 万亿预训练 tokens（大约 $32\times$ 与我们的baseline一样多).在第 3.4.1 节中，我们展示了在一些下游任务上，RealNews-like、WebText-like 和 Wikipedia + TBC 数据集的预训练优于 C4 的预训练。然而，这些数据集变体足够小，以至于在 1 万亿 tokens 的预训练过程中它们将被重复数百次。由于我们在第 3.4.2 节中表明这种重复可能有害，因此我们选择继续使用 C4 数据集。

模型大小 在第 3.6 节中，我们还展示了扩大baseline模型大小如何提高性能。但是，在可用于 fine-tuning 或推理的计算资源有限的情况下，使用较小的模型可能会有所帮助。基于这些因素，我们训练了各种尺寸的模型：

- 根据。这是我们的baseline模型，其超参数在第 3.1.1 节中进行了描述。它有大约 2.2 亿个参数。
- 小的。我们考虑一个更小的模型，它通过使用 $d_{\text{model}} = 512, d_{\text{ff}} = 2,048$、-headed attention 以及编码器和解码器中各只有 6 层来缩小baseline。这个变体有大约 6000 万个参数。
- 大的。由于我们的baseline使用 BERT $\mathbf{BASE}^{-sized}$ 编码器和解码器，我们还考虑了一种变体，其中编码器和解码器的大小和结构都与 BERTLARGE相似。具体地，这一变体使用 $d_{\text{model}} = 1,024$ ， $d_{\text{ff}} = 4,096, d_{\text{kv}} = 64, 16$-headed attention，编码器和解码器各 24 层，产生大约 7.7 亿个参数。
- 3B 和 11B。为了进一步探索在使用更大的模型时可能产生什么样的性能，我们考虑了两个额外的变体。在这两种情况下，我们都使用 $d_{\text{model}} = 1024$，一个 24 层编码器和解码器，以及 $d_{\text{kv}} = 128$。对于“3B”变体，我们使用具有 32 头注意力的 $d_{\text{ff}} = 16,384$，这产生了大约 28 亿个参数；对于“11B”，我们使用具有 128 头注意力的 $d_{\text{ff}} = 65,536$ 生成具有大约 110 亿个参数的模型。我们之所以选择扩大 d_{ff}，是因为现代加速器（例如我们训练模型所用的 TPU）对于像 Transformer 的前馈网络中的大型密集矩阵乘法最有效。

多任务预训练 在第 3.5.3 节中，我们展示了在 fine-tuning 之前对无监督和监督任务的多任务混合进行预训练以及单独对无监督任务进行预训练。这是“MT-DNN”提倡的方法（Liu et al., 2015, 2019b）。它还具有能够在整个训练期间监控“下游”性能的实际好处，而不仅仅是在 fine-tuning 期间。因此，我们在最后一组实验中使用了多任务预训练。我们假设训练时间较长的较大模型可能受益于较大比例的未标记数据，因为它们更有可能过度拟合较小的训练数据集。然而，我们也注意到第 3.5.3 节的结果表明，fine-tuning 在多任务预训练之后可以减轻一些可能因选择次优比例的未标记数据而引起的问题。基于这些想法，在使用标准样本比例混合（在第 3.5.2 节中描述）之前，我们用以下人工数据集大小替换我们的未标记数据：小型 710,000，基础 2,620,000，大型 8,660,000，3B 33,500,000，和 11B 为 133,000,000。对于所有模型变体，我们还在预训练期间将 WMT 英语到法语和 WMT 英语到德语数据集的有效数据集大小限制为 1M 个样本。

微调单个 GLUE 和 SuperGLUE 任务 到目前为止，当 GLUE 和 SuperGLUE fine-tuning 时，我们已经 concat 启用了每个基准中的所有数据集，因此我们只为 GLUE 和 SuperGLUE 建立了 fine-tune 模型一次.这种方法使我们的研究在逻辑上更简单，但我们发现与 fine-tuning 单独在任务上相比，这在某些任务上牺牲了少量性能。 fine-tuning 在单个任务上的一个潜在问题是，我们可能会很快过度适应低资源任务，否则可以通过同时对所有任务进行训练来缓解这个问题。例如，对于许多低资源 GLUE 和 SuperGLUE 任务，我们的 2^{11} 长度为 512 的大 batch 大小序列将导致整个数据集在每个 batch 中出现多次。因此，我们在 batch 期间为每个 GLUE 和 SuperGLUE 任务使用较小的 fine-

tuning 大小，即 8 个长度为 512 的序列。我们还每 1,000 步而不是每 5,000 步保存 checkpoints，以确保我们可以在过度拟合之前访问模型的参数。

Beam search 我们之前的所有结果都是使用贪心解码报告的。对于具有长输出序列的任务，我们发现使用集束搜索可以提高性能（Sutskever 等人，2014 年）。具体来说，我们对 WMT 翻译和 CNN/DM 摘要任务使用 4 的波束宽度和 $\alpha = 0.6$ 的长度惩罚（Wu 等人，2016 年）。

测试集 由于这是我们的最后一组实验，我们报告的是测试集而不是验证集的结果。对于 CNN/Daily Mail，我们使用随数据集分发的标准测试集。对于 WMT 任务，这对应于对英语-德语使用 newstest2014，对英语-法语使用 newstest2015，对英语罗马尼亚语使用 newstest2016。对于 GLUE 和 SuperGLUE，我们使用基准评估服务器来计算官方测试集分数。[15]16 对于 SQuAD，在测试集上进行评估需要在基准服务器上运行推理。不幸的是，该服务器上的计算资源不足以从我们最大的模型中获得预测。因此，我们改为继续报告 SQuAD 验证集的性能。幸运的是，在 **SQuAD** 测试集上具有最高性能的模型也报告了验证集上的结果，因此我们仍然可以与表面上最先进的模型进行比较。

除了上述变化之外，我们使用与baseline相同的训练过程和超参数（AdaFactor 优化器、预训练的平方根倒数学习率 schedule、fine-tuning 的恒定学习率、dropout 正则化、词汇表、 ETC。）。作为参考，这些详细信息在第 2 节中进行了描述

最后一组实验的结果如表 14 所示。总的来说，我们在我们考虑的 24 项任务中的 18 项上实现了最先进的性能。正如预期的那样，我们最大的（110 亿个参数）模型在所有任务的模型大小变体中表现最佳。我们的 T5-3B 模型变体在一些任务中确实击败了之前的技术水平，但是将模型大小扩展到 110 亿个参数是实现我们最佳性能的最重要因素。我们现在分析每个单独基准的结果。

我们取得了 90.3 的最先进平均 GLUE 分数。值得注意的是，对于自然语言推理任务 MNLI、RTE 和 WNLI，我们的表现明显优于之前的最新技术水平。 RTE 和 WNLI 是机器性能历来落后于人类性能的两个任务，分别为 93.6 和 95.9 (Wang et al., 2018)。就参数数量而言，我们的 11B 模型变体是已提交给 GLUE 基准测试的最大模型。然而，大多数得分最高的提交都使用大量的集成和计算来产生预测。例如，性能最佳的 ALBERT 变体（Lan 等人，2019 年）使用的模型在大小和架构上与我们的 3B 变体相似（尽管由于巧妙的参数共享，它的参数少得多）。为了在 GLUE 上产生令人印象深刻的性能，ALBERT 作者 ensemble 根据任务开发了“从 6 到 17”的模型。这可能导致使用 ALBERT ensemble 进行预测比使用 T5-11B 进行预测的计算成本更高。

对于 SQuAD，我们在精确匹配得分上比之前的最先进技术 (ALBERT (Lan et al., 2019)) 高出一分以上。 SQuAD 是三年前创建的长期基准，最近的改进仅将最先进技术提高了零点一个百分点。我们注意到，当在测试集上报告结果时，它们通常基于 ensemble 模型和/或利用外部数据集（例如 TriviaQA（Joshi 等人，2017 年）或 NewsQA（Trischler 等人，2016 年）)) 来扩充小型 SQuAD 训练集。人类在 SQuAD 上的精确匹配和 F1 指标分别估计为 82.30 和 91.22（Rajpurkar 等人，2016 年），因此尚不清楚进一步改进该基准是否有意义。

15. http://gluebenchmark.com
16. http://super.gluebenchmark.com

Model	GLUE Average	CoLA Matthew's	SST-2 Accuracy	MRPC F1	MRPC Accuracy	STS-B Pearson	STS-B Spearman
Previous best	89.4[a]	69.2[b]	97.1[a]	**93.6**[b]	**91.5**[b]	92.7[b]	92.3[b]
T5-Small	77.4	41.0	91.8	89.7	86.6	85.6	85.0
T5-Base	82.7	51.1	95.2	90.7	87.5	89.4	88.6
T5-Large	86.4	61.2	96.3	92.4	89.9	89.9	89.2
T5-3B	88.5	67.1	97.4	92.5	90.0	90.6	89.8
T5-11B	**90.3**	**71.6**	**97.5**	92.8	90.4	**93.1**	**92.8**

Model	QQP F1	QQP Accuracy	MNLI-m Accuracy	MNLI-mm Accuracy	QNLI Accuracy	RTE Accuracy	WNLI Accuracy
Previous best	74.8[c]	**90.7**[b]	91.3[a]	91.0[a]	**99.2**[a]	89.2[a]	91.8[a]
T5-Small	70.0	88.0	82.4	82.3	90.3	69.9	69.2
T5-Base	72.6	89.4	87.1	86.2	93.7	80.1	78.8
T5-Large	73.9	89.9	89.9	89.6	94.8	87.2	85.6
T5-3B	74.4	89.7	91.4	91.2	96.3	91.1	89.7
T5-11B	**75.1**	90.6	**92.2**	**91.9**	96.9	**92.8**	**94.5**

Model	SQuAD EM	SQuAD F1	SuperGLUE Average	BoolQ Accuracy	CB F1	CB Accuracy	COPA Accuracy
Previous best	90.1[a]	95.5[a]	84.6[d]	87.1[d]	90.5[d]	95.2[d]	90.6[d]
T5-Small	79.10	87.24	63.3	76.4	56.9	81.6	46.0
T5-Base	85.44	92.08	76.2	81.4	86.2	94.0	71.2
T5-Large	86.66	93.79	82.3	85.4	91.6	94.8	83.4
T5-3B	88.53	94.95	86.4	89.9	90.3	94.4	92.0
T5-11B	**91.26**	**96.22**	**88.9**	**91.2**	**93.9**	**96.8**	**94.8**

Model	MultiRC F1a	MultiRC EM	ReCoRD F1	ReCoRD Accuracy	RTE Accuracy	WiC Accuracy	WSC Accuracy
Previous best	84.4[d]	52.5[d]	90.6[d]	90.0[d]	88.2[d]	69.9[d]	89.0[d]
T5-Small	69.3	26.3	56.3	55.4	73.3	66.9	70.5
T5-Base	79.7	43.1	75.0	74.2	81.5	68.3	80.8
T5-Large	83.3	50.7	86.8	85.9	87.8	69.3	86.3
T5-3B	86.8	58.3	91.2	90.4	90.7	72.1	90.4
T5-11B	**88.1**	**63.3**	**94.1**	**93.4**	**92.5**	**76.9**	**93.8**

Model	WMT EnDe BLEU	WMT EnFr BLEU	WMT EnRo BLEU	CNN/DM ROUGE-1	CNN/DM ROUGE-2	CNN/DM ROUGE-L
Previous best	**33.8**[e]	**43.8**[e]	**38.5**[f]	43.47[g]	20.30[g]	40.63[g]
T5-Small	26.7	36.0	26.8	41.12	19.56	38.35
T5-Base	30.9	41.2	28.0	42.05	20.34	39.40
T5-Large	32.0	41.5	28.1	42.50	20.68	39.75
T5-3B	31.8	42.6	28.2	42.72	21.02	39.94
T5-11B	32.1	43.4	28.1	**43.52**	**21.55**	**40.69**

表 14：我们的 T5 变体在我们研究的每项任务上的表现。 Small、Base、Large、3B 和 11 B 分别指具有 6000 万、2.2 亿、7.7 亿、30 亿和 110 亿参数的模型配置。在每个表格的第一行，我们报告了该任务的最新技术（截至 2019 年 10 月 24 日），上标表示其来源，并在该标题的末尾列出了参考文献。除了我们使用验证集的 *SQUAD* 之外，所有结果都在测试集上报告。 [a](Lan et al., 2019) [b] (Wang et al., 2019c) [c] (Zhu et al., 2019) [d] (Liu et al., 2019c) [e] (Edunov et al. ., 2018) [f] (Lample and Conneau, 2019) [g] (Dong et al., 2019) 对于 SuperGLUE，我们大大改进了最先进的技术（平均得分为 84.6 （Liu 等人，2019c）至 88.9）。 SuperGLUE 旨在包括"超出当前最先进系统范围，但大多数受过大学教育的英语使用者可以解决的任务"（Wang 等人，2019b）。我们几乎与 89.8 的人类表现相匹配（Wang 等人，2019b）。有趣的是，在阅读理解任务（MultiRC 和 ReCoRD）上，我们的表现大大超过了人类，这表明用于这些任务的评估指标可能偏向于机器预测。另一方面，人类在 COPA

和 WSC 上都达到了 100% 的准确率，这明显优于我们模型的性能。这表明我们的模型仍然存在难以完善的语言任务，尤其是在资源匮乏的环境中。

我们没有在任何 WMT 翻译任务上实现最先进的性能。这可能部分是由于我们使用了仅英语的未标记数据集。我们还注意到，这些任务的大多数最佳结果都使用回译（Edunov 等人，2018 年；Lample 和 Conneau，2019 年），这是一种复杂的数据增强方案。低资源英语到罗马尼亚语基准的最新技术还使用了其他形式的跨语言无监督训练（Lample 和 Conneau，2019 年）。我们的结果表明，规模和英语预训练可能不足以匹配这些更复杂方法的性能。更具体地说，English to German newstest2014 集的最佳结果使用了 WMT 2018 中更大的训练集（Edunov 等人，2018），这使得与我们的结果直接比较变得困难。

最后，在 CNN/Daily Mail 上，我们获得了最先进的性能，尽管在 ROUGE-2-F 分数上仅显着提高。已经表明，对 ROUGE 分数的改进不一定对应于更连贯的摘要（Paulus 等人，2017）。此外，虽然 CNN/Daily Mail 被视为抽象摘要基准，但纯提取方法已被证明效果很好（Liu，2019）。还有人认为，以最大似然训练的生成模型容易产生重复的摘要（参见等人，2017 年）。尽管存在这些潜在问题，我们发现我们的模型确实生成了连贯且基本正确的摘要。我们在附录 C 中提供了一些非精心挑选的验证集样本。

为了取得出色的成果，T5 将我们实验研究的见解与前所未有的规模相结合。请注意，在第 3.6 节中，我们发现扩大baseline模型的预训练量或大小会产生可观的收益。鉴于此，我们有兴趣衡量我们在 T5 中引入的“非缩放”更改对其强大性能的贡献有多大。因此，我们进行了最终实验，比较了以下三种配置：首先，标准baseline模型，它在 $2^{35} \approx 34\,\mathrm{B}$ tokens 上进行了预训练；其次，baseline 训练了大约 1 万亿 tokens（即与 T5 相同的预训练量），我们称之为“baseline-1T”；第三，T5-Base。请注意，baseline-1T 和 T5-Base 之间的差异包括我们在设计 T5 时所做的“非缩放”更改。因此，比较这两个模型的性能可以让我们具体衡量我们系统研究的见解的影响。

这三种模型配置的性能如表 15 所示。与第 3.6 节中的发现一致，我们发现额外的预训练提高了baseline的性能。尽管如此，T5-Base 在所有下游任务上的表现都大大优于 baseline-1T。这表明规模并不是影响 T5 的唯一因素

模型	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
⋆ baseline	83.28	19.24	80.88	71.36	26.98	39.82	27.65
baseline-1T	84.80	19.62	83.01	73.90	27.46	40.30	28.34
T5-基地	**85.97**	**20.90**	**85.44**	**75.64**	**28.37**	**41.37**	**28.98**

表 15：T5-Base 与本文其余部分使用的baseline实验设置的性能比较。结果在验证集上报告。“Baseline-1T”指的是通过在 1 万亿 tokens（T5 模型变体使用的相同数字）而不是 $2^{35} \approx 34\,\mathrm{B}$ tokens（用于baseline）上预训练baseline模型所获得的性能.

成功。我们假设较大的模型不仅受益于它们增加的尺寸，还受益于这些非缩放因素。

4. 反馈

完成我们的系统研究后，我们首先回顾一些最重要的发现。我们的结果提供了一些高层次的观点，说明哪些研究途径可能或多或少有希望。最后，我们概述了一些我们认为可能为进一步发展该领域提供有效方法的主题。

4.1. 要点

文本到文本我们的文本到文本框架提供了一种简单的方法，可以使用相同的损失函数和解码程序在各种文本任务上训练单个模型。我们展示了这种方法如何成功应用于抽象摘要等生成任务、自然语言推理等分类任务，甚至 STS-B 等回归任务。尽管它很简单，但我们发现文本到文本框架获得了与特定任务架构相当的性能，并在与规模结合时最终产生了最先进的结果。

架构 虽然一些关于 NLP 迁移学习的工作考虑了 Transformer 的架构变体，但我们发现原始的编码器-解码器形式在我们的文本到文本框架中效果最好。尽管编码器-解码器模型使用的参数是“仅编码器”（例如 BERT）或“仅解码器”（语言模型）架构的两倍，但它具有相似的计算成本。我们还表明，在编码器和解码器中共享参数不会导致性能大幅下降，同时将总参数数量减半。

无监督目标 总的来说，我们发现大多数“去噪”目标（训练模型重建随机破坏的文本）在文本到文本设置中的表现相似。因此，我们建议使用生成短目标序列的目标，以便无监督预训练的计算效率更高。

数据集 我们引入了“Colossal Clean Crawled Corpus”(C4)，其中包含来自 Common Crawl 网络转储的启发式清理文本。将 C4 与使用额外过滤的数据集进行比较时，我们发现对域内未标记数据进行训练可以提高一些下游任务的性能。但是，限制到单个域通常会导致较小的数据集。我们分别表明，当未标记的数据集足够小以至于在预训练过程中重复多次时，性能会下降。这激发了使用像 C4 这样的大型多样化数据集来完成通用语言理解任务。

训练策略 我们发现，在 fine-tuning 期间更新所有预训练模型参数的基本方法优于旨在更新较少参数的方法，尽管更新所有参数的成本最高。我们还尝试了多种方法来同时在多个任务上训练模型，在我们的文本到文本设置中，这简单地对应于在构建 batches 时混合来自不同数据集的样本。多任务学习的主要关注点是设置每个任务的训练比例。我们最终没有找到一种策略来设置混合比例，以匹配无监督预训练后有监督 fine-tuning 基本方法的性能。然而，我们发现 fine-tuning 在对混合任务进行预训练后产生了与无监督预训练相当的性能。

缩放 我们比较了利用额外计算的各种策略，包括在更多数据上训练模型、训练更大的模型以及使用 ensemble 模型。我们发现每种方法都能显着提高性能，尽管在更多数据上训练较小的模型通常比用较少的步骤训练较大的模型表现更好。我们还展示了 ensemble 模型可以提供比单个模型好得多的结果，它提供了一种利用额外计算的正交方法。来自相同基础预训练模型的 fine-tuned 的集成模型表现比预训练差，并且 fine-tuning 所有模型完全分开，尽管 fine-tune-only 集成仍然大大优于单个模型。

突破极限 我们结合了上述见解并训练了更大的模型（多达 110 亿个参数），以在我们考虑的许多基准测试中取得最先进的结果。对于无监督训练，我们从 C4 数据集中提取文本并应用破坏 tokens 连续跨度的去噪目标。我们在 fine-tuning 单个任务之前对多任务混合进行了预训练。总体而言，我们的模型接受了超过 1 万亿 tokens 的训练。为了促进我们结果的复制、扩展和应用，我们发布了我们的代码、 C4 数据集和每个 T5 变体的预训练模型权重。 [1]

4.2. 展望

大型模型的不便性 我们研究的一个不足为奇但重要的结果是，大型模型往往表现更好。用于运行这些模型的硬件不断变得更便宜和更强大这一事实表明，扩大规模可能仍然是实现更好性能的有希望的方式（Sutton，2019 年）。然而，在某些应用程序和场景中，使用更小或更便宜的模型总是有帮助的，例如在执行客户端推理或联合学习时（Konečnỳ 等人，2015 年、2016 年）。相关地，迁移学习的一个有益用途是在低资源任务上获得良好性能的可能性。低资源任务通常（根据定义）发生在缺乏资产来标记更多数据的环境中。因此，低资源应用程序通常也只能有限地访问计算资源，这可能会产生额外成本。因此，我们提倡研究使用更便宜的模型实现更强性能的方法，以便迁移学习可以应用于影响最大的地方。目前沿着这些思路开展的一些工作包括蒸馏（Hinton 等人，2015 年；Sanh 等人，2019 年；Jiao 等人，2019 年）、参数共享（Lan 等人，2019 年）和条件计算（Shazeer 等人，2019 年）。 , 2017).

更有效的知识提取 回想一下，预训练的目标之一是（笼统地说）为模型提供通用"知识"，以提高其在下游任务上的性能。我们在这项工作中使用的方法是训练模型以对破坏的文本跨度进行去噪，这是目前常见的做法。我们怀疑这种简单的技术可能不是教授模型通用知识的非常有效的方法。更具体地说，无需首先在 1 万亿 fine-tuning 文本上训练我们的模型就能够获得良好的 tokens 性能将很有用。沿着这些方向的一些并行工作通过预训练模型来区分真实文本和机器生成的文本来提高效率（Clark 等人，2020 年）。

形式化任务之间的相似性我们观察到，对未标记的域内数据进行预训练可以提高下游任务的性能（第 3.4 节）。这一发现主要依赖于基本观察，例如 SQuAD 是使用维基百科的数据创建的。对预训练任务和下游任务之间的"相似性"形成一个更严格的概念是很有用的，这样我们就可以对使用什么来源的未标记数据做出更有原则的选择。在计算机视觉领域，沿着这些思路进行了一些早期的实证研究（Huh 等人，2016 年；Kornblith 等人，2018 年；He 等人，2018 年）。更好地理解任务的相关性也有助于选择有监督的预训练任务，这已被证明对 GLUE 基准测试有帮助（Phang 等人，2018 年）。

与语言无关的模型 我们失望地发现，仅英语预训练在我们研究的翻译任务上没有取得最先进的结果。我们也有兴趣避免需要提前指定词汇表可以编码哪些语言的后勤困难。为了解决这些问题，我们有兴趣进一步研究与语言无关的模型，即无论文本语言如何都能以良好性能执行给定 NLP 任务的模型。这是一个特别相关的问题，因为英语不是世界上大多数人的母语。

这篇论文的动机是最近关于 NLP 迁移学习的大量工作。在我们开始这项工作之前，这些进步已经在基于学习的方法尚未被证明有效的环境中实现了突破。我们很高兴能够延续这一趋势，例如通过在 SuperGLUE 基准测试中接近人类水平的性能，这是一项专门为现代迁移学习 pipeline 设计的困难任务。我们的结果源于直接和统一的文本到文本框架、我们新的 C4 数据集以及我们系统研究的见解的结合。此外，我们还提供了该领域的实证概览及其现状的观点。我们很高兴看到继续使用迁移学习来实现通用语言理解的目标。

5. 致谢

感谢 Grady Simon、Noah Fiedel、Samuel R. Bowman、Augustus Odena、Daphne Ippolito、Noah Constant、Orhan Firat、Ankur Bapna 和 Sebastian Ruder 对本手稿的评论； Zak Stone 和 TFRC 团队的支持； Austin Tarango 对数据集创建的指导； Melvin Johnson、Dima Lepikhin、Katrin Tomanek、Jeff Klingner 和 Naveen Arivazhagan 对多任务机器翻译的深入了解； Neil Houlsby 征求适配器层的意见； Olga Wichowska、Ola Spyra、Michael Banfield、Yi Lin 和 Frank Chen 在基础设施方面的协助； Etienne Pot、Ryan Sepassi 和 Pierre Ruyssen 在 TensorFlow 数据集上的合作； Rohan Anil 帮助我们下载 Common Crawl 的 pipeline； Robby Neale 和 Taku Kudo 在 SentencePiece 上的工作；以及 Google Brain 团队的许多其他成员的讨论和见解。