计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2024,33(10):163−173 [doi: 10.15888/j.cnki.csa.009666]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于神经机器翻译的模型反混淆方法①

朱　浪[1], 刘彬彬[2], 李嘉璇[1], 郑启龙[1]

[1](中国科学技术大学 计算机科学与技术学院, 合肥 230027)
[2](合肥工业大学 计算机科学与技术系, 合肥 230601)
通信作者: 郑启龙, E-mail: QLZheng@ustc.edu.cn

摘　要: 模型混淆是指将神经网络等价地转换为另一种形式, 是一种高效且低成本的神经网络保护技术. 为了发现模型混淆的缺陷, 研究人员提出了模型反混淆技术, 以期望改进模型混淆方法. 然而, 现有的模型反混淆技术研究较少, 并且适用场景和反混淆效果有限. 因此, 本文提出一种基于神经机器翻译 (neural machine translation, NMT) 技术的模型反混淆方法. 该方法将模型的反混淆任务建模成一个 seq2seq 的任务, 首先对混淆模型进行更详细的序列表示, 然后对权重参数中的混淆信息进行识别并处理, 最后再使用基于 NMT 的模型进行反混淆翻译. 实验结果表明, 该方法弥补了已有方法的不足, 能够有效地捕捉模型的混淆特征并对模型的架构进行恢复, 可以作为一种模型反混淆的通用方案.

关键词: 神经网络模型混淆; 神经网络模型反混淆; 神经机器翻译; Transformer

引用格式: 朱浪,刘彬彬,李嘉璇,郑启龙.基于神经机器翻译的模型反混淆方法.计算机系统应用,2024,33(10):163–173. http://www.c-s-a.org.cn/1003-3254/9666.html

Model Deobfuscation Method Based on Neural Machine Translation

ZHU Lang[1], LIU Bin-Bin[2], LI Jia-Xuan[1], ZHENG Qi-Long[1]

[1](School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)
[2](Department of Computer Science and Technology, Hefei University of Technology, Hefei 230601, China)

Abstract: Model obfuscation refers to the equivalent transformation of neural networks into another form, which is an efficient and low-cost technique for protecting neural networks. To detect the flaws of model obfuscation, researchers have proposed model deobfuscation techniques in the hope of improving model obfuscation methods. However, model deobfuscation techniques are not fully explored, with limited applicability and effectiveness. Therefore, this study proposes a model deobfuscation method based on neural machine translation (NMT). This method models a deobfuscation task as a seq2seq task. It provides a more detailed sequential representation of the obfuscated model, identifies and processes the obfuscated information in the weight parameters, and utilizes an NMT-based model for deobfuscation translation. The experimental results demonstrate that this method addresses the shortcomings of existing methods, effectively capturing the obfuscation features and restoring the architectures of models. It can serve as a general solution to model deobfuscation.

Key words: neural network model obfuscation; neural network model deobfuscation; neural machine translation (NMT); Transformer

近些年, 深度学习作为机器学习的一个分支, 不管是在计算机视觉, 还是自然语言处理等相关领域, 都得到了广泛的应用. 然而, 深度神经网络 (deep neural network, DNN) 模型作为一种有价值的知识产权[1], 也

① 收稿时间: 2024-04-15; 修改时间: 2024-05-14; 采用时间: 2024-05-23; csa 在线出版时间: 2024-09-02
CNKI 网络首发时间: 2024-09-03

面临着各种安全风险. 下面是两种典型的风险.

首先, 模型面临的第 1 个安全风险是存在被窃取的风险. 模型窃取[2]是指攻击者通过各种手段来获取目标模型的关键信息的过程, 这些信息主要包括模型的架构与权重参数等, 一旦获取了这些信息, 攻击者便可以轻松地重构出相似或相同功能的模型. 这种攻击可能会导致严重的安全风险和隐私泄露问题, 当目标模型涉及敏感数据或商业机密时, 将对模型的所有者造成极大的影响.

除了面临被窃取的风险外, 一个神经网络模型还面临着遭受对抗攻击[3]的风险. 对抗攻击是指攻击者精心构造一些对抗性的例子对模型进行欺骗, 而对对抗性例子的脆弱性是在安全关键场景中应用 DNN 的主要风险之一[4]. 例如, 在交通标志识别系统中构造停车标志或者在物体识别系统中去除行人的分割等操作, 能够误导自动驾驶车辆造成严重的后果. 并且, 已有的研究表明, 精确的原模型架构信息能够使对原模型的攻击成功率提高好几倍[4], 因此, 在对模型进行对抗攻击之前, 也会尽可能地获取模型的相关信息. 总的来说, 揭示模型内部特征信息的模型提取攻击是 DNN 系统中重要的攻击模型之一.

为了应对上述风险, 研究人员提出了不同的解决方案. 模型混淆[5]因其方便实用和更小的性能开销等优点, 成为目前广泛采用的模型保护技术. 然而, 尽管模型混淆方法能够对模型进行保护, 但保护方法自身仍可能存在潜在的安全漏洞等问题. 也正因为如此, 对模型反混淆技术的研究开始出现, 其用于帮助评估模型对逆向工程和攻击的抵抗能力, 识别模型混淆方法中潜在的安全漏洞与风险, 进而改善模型混淆方法.

如上所述, 对模型反混淆技术的研究具有重要的意义. 但是, 目前关于模型的反混淆研究甚少, 并且已有方法的适用场景与反混淆效果有限. 本文的主要贡献是: 提出了一种新的基于神经机器翻译 (neural machine translation, NMT)[6]的模型反混淆方法. 旨在扩充现有模型反混淆方法的研究与探索现有模型混淆方法存在的不足, 以更好地保护神经网络模型.

本文第 1 节介绍模型反混淆的相关工作. 第 2 节介绍本文所使用的神经机器翻译技术. 第 3 节是本文模型反混淆方法的整体流程. 第 4 节是相关的实验与分析. 最后是本文的结论与展望.

1 相关工作

当前更多的研究集中在了模型的混淆保护方法上, 对于模型反混淆的研究甚少, 截至 2024 年 3 月, 只有文献[7]提出了一种模型反混淆方法.

1.1 模型混淆

模型混淆[5]是指保证推理结果不变的前提下, 通过修改模型的架构和参数等信息, 对模型的计算逻辑进行修改与隐藏, 达到对模型进行保护的目的.

Li 等人[1]提出了一种先进的神经网络模型混淆器 NeurObfuscator, 里面主要使用了 6 大类混淆转换[8–11]对模型进行混淆, 使得混淆之后的模型拥有与原模型不同的运行时架构提示信息, 并且为了使基于侧信道的架构窃取 (side-channel-based architecture stealing, SCAS) 攻击[12–15]方法尽可能地失效, 还采用了遗传算法[16]来组合上述的混淆转换. 事实上, 由于直接改变了模型的架构和参数, 即使攻击者直接窃取到了混淆后的模型也难以理解. 然而, ObfuNAS[11]表明上述混淆方法存在无效混淆的风险, 由于最终混淆的架构往往仍然可以训练, 将使得攻击者可以获得比受害者模型甚至更高的准确性模型, 这与混淆背道而驰. 在此基础上, ObfuNAS 通过模拟攻击者的方式, 利用获得的结果以及遗传算法去指导选择最适合的混淆架构, 最终将提取的架构的模型精度最小化. 其不仅能保持受害者模型的原始推理精度, 也能有效地防止攻击者去训练一个竞争性的替代模型. 除此之外, 其他的一些工作[17,18]则提出了一些不同的混淆思路, 它们与通常的模型混淆方法不同, 虽然能保护模型, 但过程相对复杂且繁琐, 本文忽略这些方法.

1.2 6 大类混淆转换

上述的一类先进混淆工具以 6 大类混淆转换为核心, 它们基于函数保持[8]的概念, 对模型进行等价转换. 下面本文对该 6 类转换进行详细介绍, 所使用到的主要符号及其表示意义如表 1 所示.

(1) 层分支. 层分支操作可以将一个节点切分成许多个小节点, 以改变原网络的拓扑结构. 以卷积操作为例, 假设权重为 $W_{oc,ic,kh,kw}$, 则可以把卷积核的权重按不同的通道数进行切分, 得到新的权重 $W_{oc_1,ic,kh,kw}$, $W_{oc_2,ic,kh,kw}$, $\cdots$, 接着将得到的不同数量通道的权重作为一个新的卷积核的权重, 最后再将各个新的卷积操作的计算结果拼接 (concat) 起来, 结果保持一

致. 即:

$$XW_{oc,ic,kh,kw} = Concat(XW_{oc_1,ic,kh,kw}, XW_{oc_2,ic,kh,kw}, \cdots) \tag{1}$$

表 1 主要符号及其表示意义

符号	意义
$X^{(i)}$	第i层的输入
$W^{(i)}$	第i层的权重
$E^{(i)}$	第i层的单位权重
O	全0张量
φ,σ	激活函数/ReLU激活函数
n,ε	批次大小/接近0的数
ic,oc	输入通道/输出通道大小
kh,kw	Conv2D的Kernel尺寸
ih,iw	输入高度/宽度大小
oh,ow	输出高度/宽度大小

(2) 伪添加. 伪添加操作通过在一些激活结果后面添加相同形状的 0 张量, 对网络架构进行了微小的改变, 是一种简单实用的混淆转换.

$$X_{n,oc,oh,ow} + O_{n,oc,oh,ow} = X_{n,oc,oh,ow} \tag{2}$$

(3) 层加深. 层加深操作通过在一些层的开头或末尾插入一些恒等的计算层, 以达到对模型层加深的目的. 在实际应用中, 常见的是插入恒等的卷积层或者全连接层. 为了保证层的完整性, 还需要选择适当的激活函数. 在当前的网络模型中, ReLU 激活函数因具有不错的效果而被很多研究人员所选择, 对混淆而言, 更重要的是, ReLU 激活函数有一个良好的性质: 幂等性, 这为添加完整的混淆层提供了基础.

$$\sigma(\sigma(XW)E) = \sigma(XW) \tag{3}$$

(4) 层跳过. 层跳过操作在一些层的开头或末尾添加一个 0 计算层并与之结果相加, 使模型拥有了一个残差结构. 常用的 0 计算层一般是 0 卷积计算层, 其卷积核用的权重参数全为 0.

$$X_{n,ic,ih,iw} + \varphi(X_{n,ic,ih,iw}W) = X_{n,ic,ih,iw} \tag{4}$$

(5) 层加宽. 层加宽操作通过复制已有通道的参数加入计算, 达到对该卷积层加宽的目的. 例如, 若模型第i层卷积层与第$i+1$层卷积层对应的权重分别是$W^{(i)}_{oc,ic,kh,kw}$与$W^{(i+1)}_{oc^{(i+1)},oc^{(i)},kh,kw}$, 则可以对第 1 个权重的输出通道进行复制, 不妨通道 1 复制 2 份, 以及通道 2 复制 1 份, 此时第i层权重变为$W^{(i)}_{oc+3,ic,kh,kw}$, 然后再对第$i+1$层的输入通道进行类似的复制, 对应权重变为$W^{(i+1)}_{oc^{(i+1)},oc+3^{(i)},kh,kw}$, 最后把第$i+1$层对应的通道权重值根据复制的情况进行对应比例缩小, 最终计算的结果将与加宽分解前的结果是一致的.

(6) 核加宽. 核加宽操作通过对卷积核填充 0 的形式, 改变了卷积核的形状. 同时, 为了保持输出结果与原结果完全一致, 还需要对应地修改原卷积操作的 padding 参数.

以上便是主要的 6 大类混淆转换, 分别对应了图 1 中的 (1)–(6). 另外, 在实际的使用中, 还会对相关权重参数添加噪声, 以加强混淆弹性. 添加噪声的方式为:

$$W_{obf} = W + \varepsilon,\ \varepsilon \approx 0.0 \tag{5}$$

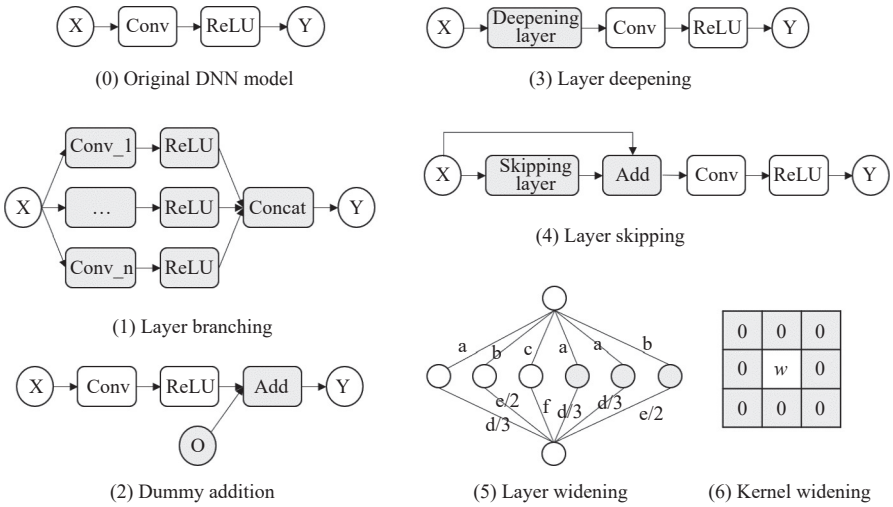

图 1 6 大类混淆转换[8–11]

1.3 模型反混淆

模型反混淆与模型混淆相对应, 其通过各种方法对模型的架构等信息进行恢复, 是一种特殊的模型窃取攻击, 通常以白盒攻击的形式进行.

Ahmadi 等人[7]针对 NeurObfuscator 混淆工具, 提出了一种基于学习的反混淆方法. 在 NeurObfuscator 混淆工具中, 为了更有效地应对 SCAS 攻击, 它采用了遗传算法来对各个混淆转换进行组合, 遗传算法的目标是使评估模型恢复情况的指标层错误率 (layer error rate, LER)[1]达到最大. 然而, 这也使得对模型的混淆过程有了确定性的混淆模式[7]. 如图 2 所示, Ahmadi 等人针对这一确定性, 将混淆前后的模型建模成层序列, 先利用对应的模型混淆工具建立数据集, 接着使用一个基于 LSTM 的 NMT 模型[19]去学习前后序列的对应关系, 能够对模型进行一定恢复. 并且更进一步地, 提出了改进后的混淆工具 ReDLock, 主要是把组合混淆转换的方法由遗传算法修改成了随机组合的算法, 打破混淆前后模型序列的确定性对应关系, 以应对 Neuro-Unlock 的反混淆方法.

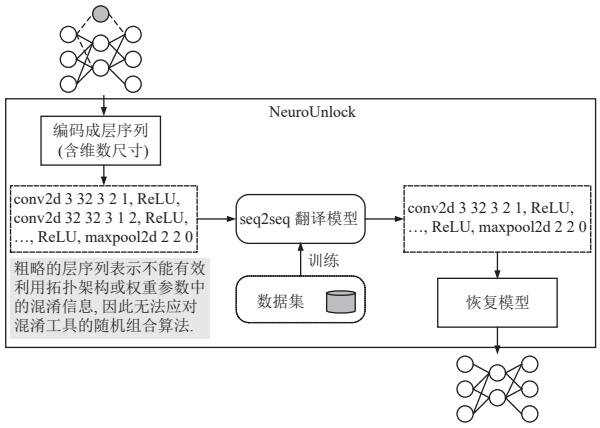

图 2 NeuroUnlock 对混淆模型的恢复过程

总的来说, 现有的一类先进模型混淆工具以 6 大类基础混淆转换为核心, 在不同的场景中还会使用不同的算法去组合这些混淆转换, 并且当使用随机算法进行组合时, 能够使得固有的混淆模式消失, 导致已有的模型反混淆方法无法对其进行有效的反混淆. 为此, 本文围绕恢复模型混淆前后的序列对应关系, 对整个反混淆流程进行了优化. 首先对混淆后模型进行更详细的序列表示, 然后设计混淆信息提取模块对混淆信息进行提取并处理, 最后再将得到的混淆信息用于辅助后续的神经机器翻译模型完成反混淆翻译任务, 能够有效地应对上述模型混淆方法.

2 神经机器翻译技术

机器翻译是自然语言处理任务中的一个经典的子领域, 主要研究如何将文本或语音从一种语言翻译到另一种语言. 在它的发展历史中主要有 3 个发展阶段, 分别是基于规则的机器翻译、统计机器翻译和神经机器翻译 NMT, 而 NMT 由于其简单的架构和善于捕捉句子长依赖性等优势[20], 逐渐成为机器翻译的主流技术.

一个典型的神经机器翻译流程如图 3 所示. 现代 NMT 模型一般由编码器与解码器两部分组成, 它们将源语句序列转换为对应的目标语句序列. 而按照模型架构的不同类别, 目前取得良好性能的 NMT 模型可以分为 3 类, 分别是基于 CNN 的 NMT、基于 RNN 的 NMT 与基于注意力的 NMT, 它们的代表模型分别为 ConvS2S[21]、RNMT+[22]与 Transformer[23].

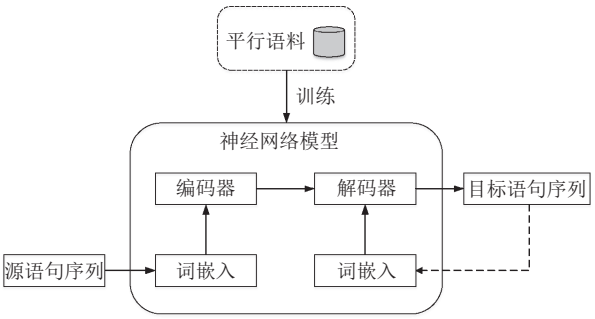

图 3 神经机器翻译工程流程

通常地, 一个模型可以进行序列化表示. 因此, 模型混淆实际上是将一个序列转变成另一个序列的过程. 倘若模型混淆过程中存在固有的混淆模式, 就会导致两个序列存在确定的对应关系, 此时, 便可以利用神经机器翻译技术来完成反混淆翻译.

3 本文提出的模型反混淆方法

3.1 威胁模型

本文对混淆后的模型进行攻击, 对其恢复之后得到原模型的计算序列. 在这个过程中, 只关注部署在边缘设备上的模型, 假设攻击者对受害者的 DNN 架构、参数、训练算法或超参数没有先验知识, 并且攻击者可以进行以下操作.

(1) 通过直接提取或 SCAS 攻击等方法获取混淆后模型的完整信息, 包括架构和参数等. 获取这些信息后将攻击场景变为了白盒攻击的形式, 然后利用这些信息, 可以将混淆模型使用 PyTorch 实现.

(2) 能够以黑盒的方式访问对模型进行混淆的混

淆工具. 利用该混淆工具, 可以产生反混淆所需要的数据集.

3.2 对混淆后的 DNN 模型进行反混淆翻译

本文将模型的反混淆任务建模成 seq2seq 的反混淆翻译任务, 其总体工作流程如图 4 所示. 混淆后的模型先接入 MLIR 编译框架[24]得到对应的 torch-mlir 序列表示, 然后再经过反混淆翻译得到原模型的计算序列. 其中, 计算序列是一个模型简单的序列化表示, 它由模型主要的计算操作及其对应的维数尺寸组成, 能够准确地映射到模型的网络架构. 例如, LeNet-5 的计算序列如表 2 所示. 接下来, 将对整个过程进行详细介绍.

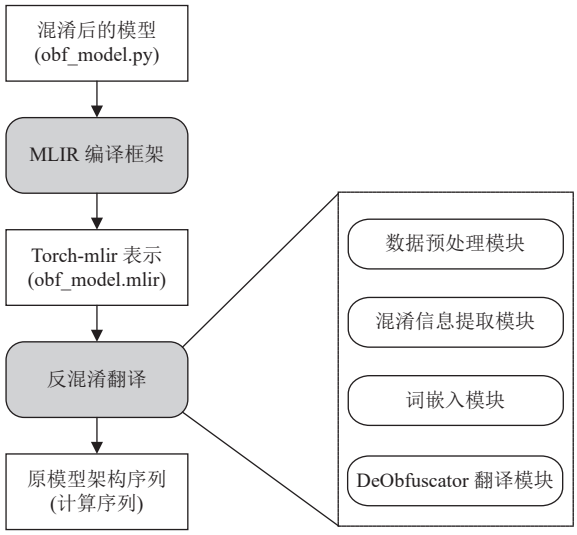

图 4 反混淆工作流程

表 2 LeNet-5 的计算序列

序列中的位置	计算操作及其维数尺寸
1–5	torch.aten.convolution, 6, 1, 5, 1
6	torch.aten.relu
7–9	torch.aten.max_pool2d, 2, 2
10–14	torch.aten.convolution, 16, 6, 5, 1
15	torch.aten.relu
16–18	torch.aten.max_pool2d, 2, 2
19	torch.aten.view
20	torch.aten.transpose.int
21, 22	torch.aten.mm, 120
23	torch.aten.add.Tensor
24	torch.aten.relu
25	torch.aten.transpose.int
26, 27	torch.aten.mm, 84
28	torch.aten.add.Tensor
29	torch.aten.relu
30	torch.aten.transpose.int
31, 32	torch.aten.mm, 10
33	torch.aten.add.Tensor

注: 这里激活函数使用ReLU, 且该计算序列的长度为33

3.2.1 更详细的序列表示

已有的反混淆方法将混淆后的模型简单表示为层序列, 会丢失模型拓扑架构与权重参数中的混淆信息. 此时, 若模型混淆工具使用随机算法的方式来选择混淆转换对模型进行混淆, 如图 5 所示, 会出现不同模型架构混淆到同一架构的情况, 而该方法无法对它们进行区分, 导致无法进行有效的反混淆工作. 因此, 本文的方法对混淆模型进行了更详细的序列表示. 具体来说, 本文借助 MLIR 编译框架, 将 torch 脚本代码实现的模型编译为了 torch-mlir 表示. 该表示使得后续流程能够充分使用模型拓扑架构与权重参数中的混淆信息, 它们能够恢复模型混淆前后的序列对应关系, 进而使得后续流程能对混淆模型进行更准确的反混淆翻译.

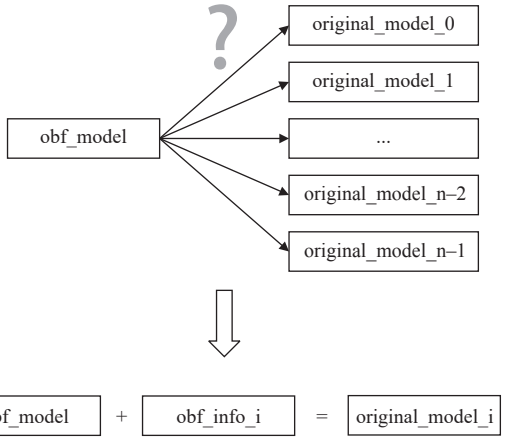

图 5 使用混淆信息恢复模型序列对应关系

3.2.2 对混淆模型进行反混淆翻译

在得到混淆模型的 torch-mlir 表示后, 需要进一步使用 NMT 技术对其进行反混淆翻译. 该翻译流程一共包括数据预处理模块、混淆信息提取模块、词嵌入模块与翻译模块 4 个部分.

(1) 数据预处理模块

在混淆模型的 torch-mlir 表示中, 由于含有冗余信息, 导致序列的长度加大, 这不利于后续的翻译工作. 故本文在尽可能不丢失关键信息的同时, 进一步将无关冗余信息进行去除. 除此之外, 为了减少词汇表的大小, 还对操作 (operation)[20]结果的名字进行了统一的编号处理. 图 6 为一个模型的 torch 脚本实现及其预处理后的表示示例. 其中, 一个模型由多个操作组成, 而一个操作则主要由结果编号、操作名 (类型) 以及拓扑参数或值信息组成.

```
class testNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv = nn.Conv2d(1, 2, 5)
        self.fc = nn.Linear(2 * 12 * 12, 10)

    def forward(self, x):
        # input shape: [1, 1, 28, 28]
        x = self.conv(x)
        x = F.relu(x)
        x = F.max_pool2d(x, (2, 2), stride=(2, 2))
        x = torch.flatten(x, 1)
        x = self.fc(x)

        return x
```

```
'Op0', 'torch.vtensor', '1', '1', '28', '28',
'Op1', 'torch.constant.int', '0',
...,
'Op9', 'torch.vtensor.literal', '0.199265048', '-0.131629825', ..., '-0.114113286', 'torch.vtensor',
'2', '1', '5', '5',
...,
'Op14', 'torch.aten.convolution', 'Op0', 'Op9', 'Op8', 'Op11', 'Op12', 'Op11', 'Op5', 'Op13', 'Op2',
'Op15', 'torch.aten.relu', 'Op14',
'Op16', 'torch.prim.ListConstruct', 'Op10', 'Op10',
'Op17', 'torch.aten.max_pool2d', 'Op15', 'Op16', 'Op16', 'Op12', 'Op11', 'Op5',
'Op18', 'torch.prim.ListConstruct', 'Op2', 'Op4',
'Op19', 'torch.aten.view', 'Op17', 'Op18',
'Op20', 'torch.aten.transpose.int', 'Op7', 'Op1', 'Op2',
'Op21', 'torch.aten.mm', 'Op19', 'Op20',
'Op22', 'torch.aten.add.Tensor', 'Op21', 'Op6', 'Op3',
'return', 'Op22', 'torch.vtensor', '1', '10'
```

图 6　模型的 torch 脚本实现及其预处理后的表示示例

(2) 混淆信息提取模块

在化简后的类 torch-mlir 序列中, 每一层的权重参数对应着不同的“torch.vtensor.literal”操作, 并且这些操作中含有数量巨大的权重参数, 这会导致无法将这些参数分别作为具体的词进行处理.

然而, 在混淆工具对模型进行混淆转换时, 会将混淆信息隐藏在这些权重参数中. 混淆工具对模型权重参数的改变主要有两种: 一种是对已有的权重参数进行修改, 例如层加宽操作通过复制已有通道的方式来对卷积层进行加宽; 另一种则是直接构建新的权重参数, 这些权重参数与混淆层相对应, 它们多为构造的恒等转换, 里面含有不少的“0 元素”与“1 元素”特征. 由此可见, 上述的权重参数在组织上或数值上存在着与混淆转换相对应的特征, 而这些数据的结构与图像数据类似, 可以利用基于 CNN 的模型进行相关特征的提取. 并且, 虽然混淆工具在混淆时添加了噪声, 但卷积神经网络中的卷积层与池化层能够有效应对这些噪声的干扰.

因此, 本文设计了一个如图 7 所示的基于 CNN 的混淆信息提取模型, 用于提取权重参数中的混淆信息. 该模型完成一个二分类任务: 判断给出的权重参数是否含有混淆信息. 如图 8 所示, 若对应的权重参数中含有混淆信息, 则将该权重参数替换为一个“obf_info_flag”单词并置于对应操作的末尾, 否则直接去掉对应的权重参数. 同时, 为了更好地分离混淆特征, 数据输入时将会进行逐元素取绝对值处理. 除此之外, 输入数据还将会被进行填充 0 或截断操作, 并统一 reshape 为 (3, 224, 224), 该大小可以覆盖数据集中绝大部分层的权重张量. 经过该模块处理后, 带有混淆信息的权重参数被转换为了一个词, 以供后续的模块继续处理.

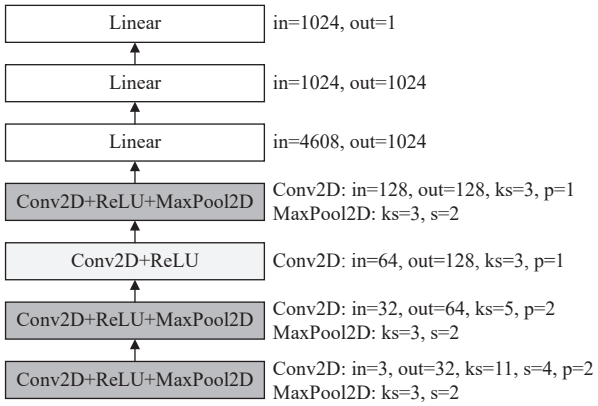

图 7　混淆信息提取模型

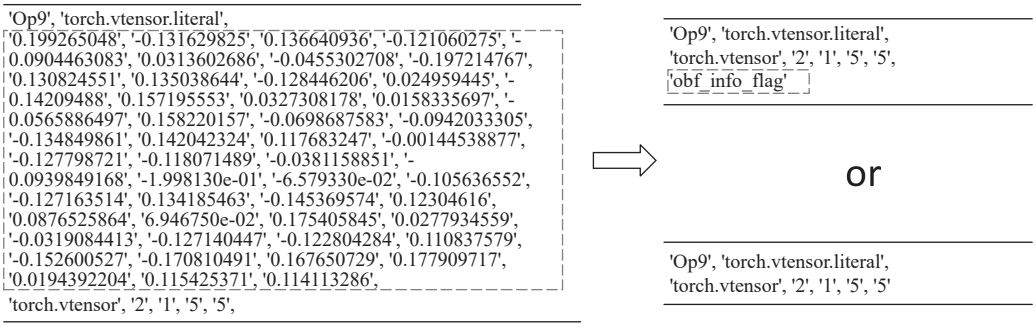

图 8　混淆信息转换

(3) 词嵌入模块

一个通常的词嵌入过程主要包括分词、语言模型训练以及从语言模型中提取词向量 3 个过程. 对于本文的反混淆翻译任务, 数据集中的模型的表示经过前

面的处理后, 已完成了分词操作, 并且词汇表主要由编号、计算操作名称以及常见的维数尺寸大小构成, 总共词数不到 600 个. 而新的表示不仅十分规整, 并且也不存在未登录词 (out of vocabulary, OOV) 与一词多义等问题. 因此, 本文直接选择统一使用常用的 Word2Vec[25] 方法中的 skip-gram 模型, 在类 torch-mlir 表示的语料库中对数据集中的词进行词嵌入. 其中, 嵌入维度大小为 64, n-gram 参数取 3. 除此之外, 位置编码则与 Transformer 中的一致.

(4) 基于 Transformer 的翻译模块

与简单的层序列表示相比, 本文基于 torch-mlir 的序列表示增加了更详细的拓扑信息与权重参数信息, 使得混淆模型的序列长度增加了很多. 而 Transformer 已经在各种翻译中取得了良好的性能, 并且, 它独特的模型架构设计使得可以并行训练以加快训练速度, 因此本文选择了基于 Transformer 的 NMT 模型来完成翻译任务. 其对处理后的混淆模型表示序列进行翻译, 得到原模型的计算序列.

Transformer 的核心思想是完全基于自注意力机制, 如图 9 所示, 其架构主要由编码器与解码器构成. 其中, 编码器部分主要包括多头注意力层、前馈神经网络层与 Add & Norm 层; 解码器部分与编码器部分大体相同, 不同的地方是解码器部分多了一个带有掩码的多头注意力层. 这是为了确保在训练过程中解码器不会查看未来的信息, 因此设计了一个注意力掩码来限制模型的访问.

另外, Transformer 模型虽然可以进行并行训练, 但其在推理时仍然同通常的 seq2seq 模型类似, 逐个词的进行生成. 在实际的实现中, 可以通过贪心算法和 beam 搜索[26]等来进行下一个词的选择.

3.3 数据集的构建

本文涉及的数据集一共有两个, 分别是混淆张量数据集与混淆模型数据集. 其中, 混淆张量数据集用来训练和测试模型对权重张量中的混淆信息识别情况, 混淆模型数据集用来对该反混淆方法进行最终的训练与测试.

由于当前的混淆工具主要面向 CNN 模型, 因此本文实现了一个随机的 DNN 模型生成工具, 该工具先生成不同的卷积层、残差连接层[27]与 inception 层[28]等经典 CNN 模型层结构, 然后再将它们组合成不同的神经网络模型. 具体来说: 对于卷积层, 卷积核尺寸从{1×1, 3×3, ⋯, 11×11}中随机选取, 其输出通道数从{6, 8, 16, 24, 32, 40, 48, 64, 128, 256}中随机选取, 残差连接层中的卷积层也类似, 在卷积层后面, 进一步随机选取不同的池化层进行池化. 除此之外, inception 层则与 GoogLeNet 中一致, 全连接层的维度则从{16, 32, 64, 128, 256}中随机选取. 整个模型的架构以卷积层和全连接层为主, 然后再以一定的概率将不同位置的卷积层替换成残差连接层或简单的 inception 层.

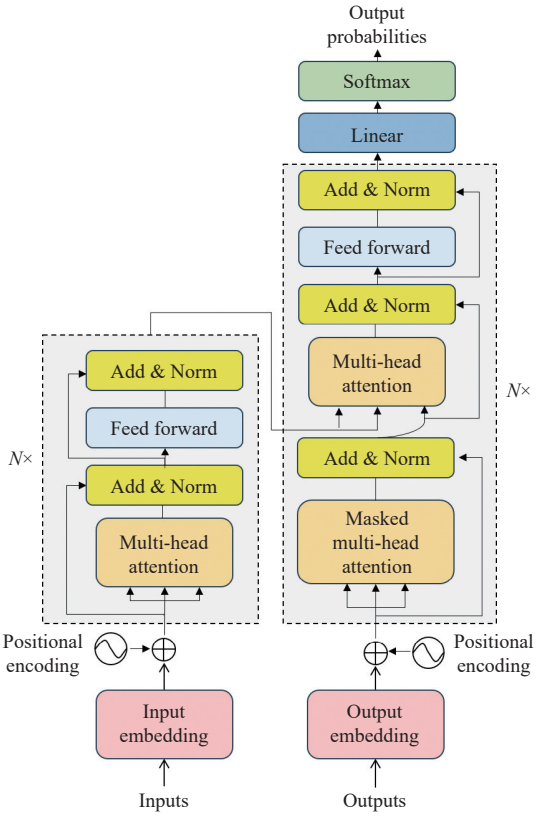

图 9 Transformer 模型架构[25]

数据集的生成流程如图 10 所示, 首先利用随机 DNN 模型生成工具生成不同的模型, 然后对这些模型的权重参数进行随机初始化, 用于模拟现实中可能出现的各种权重参数; 最后利用已有的混淆工具对初始化后的模型进行混淆, 得到混淆模型. 经上述流程得到的混淆前后的模型是一一对应的, 混淆模型及其对应的原模型构成了一个数据点, 所有数据点构成了本文的混淆模型数据集 (Dataset1).

对于混淆张量数据集 (Dataset2), 可以按照类似的方式生成. 这是由于混淆工具对模型进行混淆时, 会添

加或修改原模型的权重参数, 因此可以将混淆前的权重参数与混淆后的权重参数作对比, 若是新出现的权重张量, 则将其标记为混淆张量, 否则将其标记为非混淆张量. 具体的生成算法如算法 1 所示.

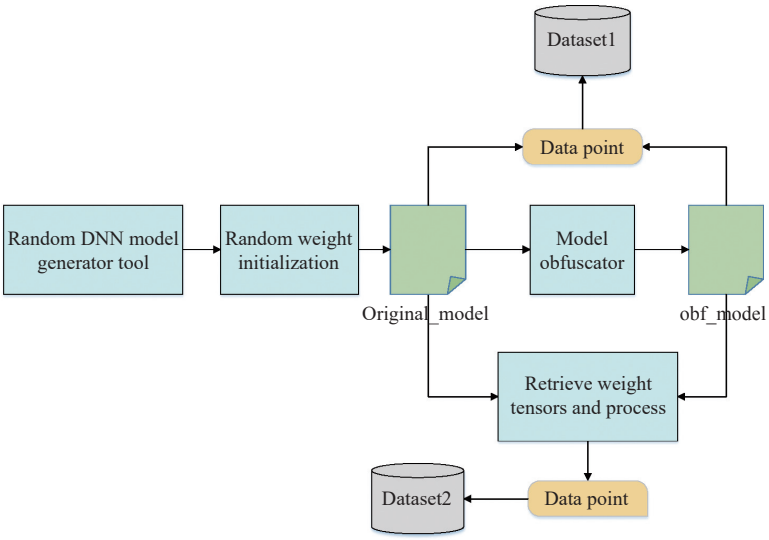

图 10 数据集生成流程

算法 1. 构建混淆信息提取模型的数据集

1) 初始化 data_list, obf_tensor_list, original_tensor_list.
2) 从 DNN 模型集合中取出一个模型 original_model, 并使用混淆工具对其混淆得到 obf_model.
3) 提取 obf_model 中的所有权重参数张量至 obf_tensor_list; 提取 original_model 中的所有权重参数张量至 original_tensor_list.
4) 逐个检查 obf_tensor_list 中的 tensor_i; 如果 tensor_i 存在于 original_tensor_list, 则将[tensor_i, 0]加入 data_list, 否则将[tensor_i, 1]加入 data_list.
5) 将该轮的 original_model 从 DNN 模型集合中删除, 清空 obf_tensor_list 与 original_tensor_list.
6) 若 DNN 模型集合非空, 则转步骤 2), 否则转步骤 7).
7) 根据实际情况将 data_list 中的张量统一填充或截断为相同的形状后终止.

4 实验与评估

本文实验环境如下: 硬件方面, CPU 为 AMD Ryzen 5800, 另外还使用了一块 NVIDIA GeForce GTX-1650 GPU; 软件方面, 操作系统为 Ubuntu 22.04.2, MLIR 对应的 LLVM 版本为 17.0.0, 深度学习框架及其版本为 Torch 1.13.1+Cu116.

实验总共分为两部分, 分别是混淆信息提取实验与模型反混淆效果检验实验.

4.1 混淆信息提取实验

为了说明混淆后权重参数中存在混淆信息并且能被准确提取, 本文利用混淆工具自制了数据集并进行了实验, 具体如下.

(1) 数据集

利用第 3.3 节中的算法 1, 为混淆张量数据集一共生成了近 20 000 条数据. 其中, 正负样本比例为 2:1.

(2) 实验设置

将数据集按照 6:2:2 的比例划分为训练集、验证集与测试集, 对第 3.2 节中的混淆信息提取模型一共训练了 100 个 epoch. 其中: 优化器采用 Adam, 学习率为 0.001, weight_decay 为 0.000 5; batch_size 为 16; 损失函数为 BCEWithLogitsLoss; 另外, 还进行了梯度裁剪操作, clip 参数取 1.0.

(3) 实验结果与分析

使用准确率 (accuracy, *Acc*) 来评估混淆信息提取模型的性能, 将准确率定义为预测正确的权重参数张量数量与总的权重参数张量数量的比值:

$$Acc_{tensor} = \frac{tensor_{\text{correct}}}{tensor_{\text{all}}} \tag{6}$$

图 11 展示了训练过程中训练集和验证集的准确率随着训练轮数增加变化的情况. 模型收敛之后, 在训练集与验证集上的准确率分别为 0.93 与 0.90 左右, 并且最终在测试集上的准确率约为 0.91.

上述实验数据说明了混淆工具对模型进行混淆后, 对应的混淆信息确实被保留在了权重参数中, 并且能够利用上述基于 CNN 的模型有效地进行提取.

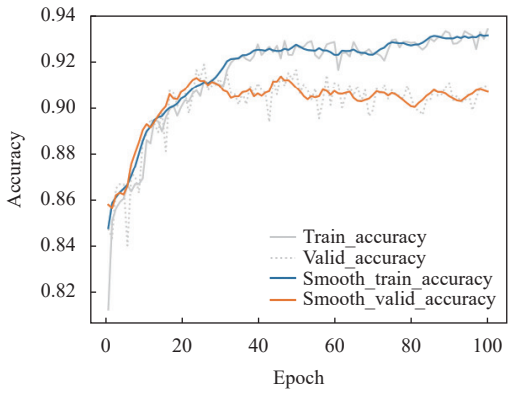

图 11 混淆信息提取模型学习过程

4.2 模型反混淆实验

为了验证本文反混淆方法的可行性和有效性, 进行了该模型反混淆实验. 需要注意的是, 与类似工作 NeuroUnlock 相比, 由于缺乏公开的数据集, 各自生成的数据集不一致, 导致它们的结果不好直接比较. 因此, 本文通过增加对照组实验的方式, 对本文所加入的关键步骤的有效性进行验证, 并将二者的结果作为本文方法与已有方法的比较. 增加的对照组除了对混淆模型进行了更详细的表示之外, 其余整体流程与 NeuroUnlock 中类似. 整个实验具体如下.

(1) 数据集

利用第 3.3 节所述的方法为混淆模型数据集一共生成了 6 000 条数据. 为了方便实验, 本文将数据集中原始模型的深度限制在了 4–10 层之间, 而混淆后模型的深度则在 22 层以内, 以加快翻译模型的收敛速度. 需要说明的是, 虽然只是在该小规模数据集上验证上述方法的可行性与有效性, 但该方法可直接移植到对应的更大的数据集上.

(2) 实验设置

将数据集按照 8:2 的比例划分为训练集与测试集, 然后对基于 Transformer 的 NMT 翻译模型训练了 50 个 epoch. 其中, 模型的编码器和解码器的个数均为 2, 多头注意力参数取为 4, 前馈层参数取为 1 024; 优化器采用 Adam, 学习率为 0.001, weight_decay 为 0.000 5; batch_size 取 4; 损失函数为 CrossEntropyLoss; 梯度裁剪操作参数 clip 取 1.0. 除此之外, 生成目标序列时, 实验组与对照组方法均采用了贪心算法, 直接选取概率最大的下一个词.

(3) 实验结果与分析

文献[7]使用了 LER 来对模型的性能进行评估, 在本文的评估中, 反混淆翻译最终生成的是原模型的计算序列表示, 因此改用计算的序列错误率 (sequence error rate, *SER*) 来评估反混淆方法的性能, 将其定义为:

$$SER = \frac{SED(Seq_{\text{recovered\_model}}, Seq_{\text{original\_model}})}{len(Seq_{\text{original\_model}})} \tag{7}$$

其中, *SED* (sequence edit distance) 指序列编辑距离[29], 它是使得两个序列相等的最少复制、删除或替换操作次数. 由于计算序列能准确地映射到模型的网络架构, 因此 *SER* 能够很好地反映反混淆方法的性能.

图 12 展示了两种方法的训练 loss 曲线. 其中, 横轴为训练的 epoch 次数, 纵轴为每个 epoch 中所有训练数据的平均 loss. 随着 epoch 次数的增加, 两种方法的平均 loss 逐渐稳定, 最终分别收敛到了 0.14 与 0.11 左右, 这说明模型能够对混淆前后序列的对应关系进行有效学习. 并且, 在训练过程中, 没有进行混淆信息提取处理的方法的平均 loss 只是略高于进行混淆信息提取处理的方法.

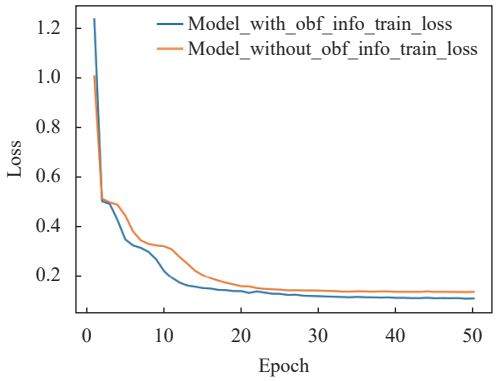

图 12 两种方法的 loss 曲线

图 13 为两种方法在测试集上的结果, 二者的平均 *SER* 分别为 0.534 与 0.316. 两者训练时的平均 loss 虽然接近, 但是二者在测试集上的结果却体现出了较大的差距. 这是因为模型在推理时, 词是逐一生成的, 由于存在不同模型架构混淆到同一架构的情况, 当没有进行混淆信息提取处理操作时, 模型会无法正确地获取下一个词, 导致后续的子序列继续出错.

另外, 虽然与类似工作 NeuroUnlock 的结果不便直接比较, 但这里也给出 NeuroUnlock 中的结果数据作为参考. 当混淆工具使用随机算法代替遗传算法时, NeuroUnlock 的平均 LER 由 0.300 增加到了 0.640, 而本文方法面对混淆工具中的随机组合算法时, 平均

SER 为 0.316. 并且, 计算序列去除维数尺寸后即为简单的层序列表示, 此时, 本文方法对应的平均 LER 仅为 0.149.

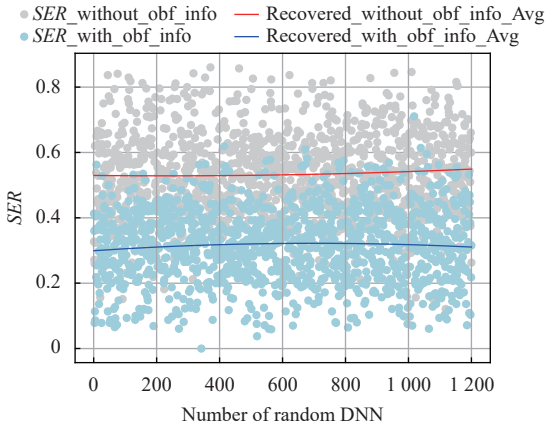

图 13 两种方法在训练集上的结果对比

上述实验数据说明了本文的反混淆方法是可行且有效的. 在本文的数据集中, 其平均 *SER* 仅约为对照组的 59.2%. 并且进一步参考 NeuroUnlock 的实验结果也可以发现, 本文的反混淆方法达到了较高的反混淆准确率. 总的来说, 本文的方法通过对混淆模型进行更详细的序列表示以及在此基础上进行的混淆信息提取处理操作, 一定程度上恢复了模型混淆前后的序列对应关系, 因而能够应对上述的一类使用随机算法组合基础混淆转换的模型混淆工具, 对混淆模型进行较为准确的反混淆恢复.

5 结论与展望

本文将模型的反混淆任务建模成一个 seq2seq 的翻译任务, 主要通过对混淆模型进行更详细的序列表示、混淆信息提取处理操作与 NMT 技术完成整个反混淆过程, 并自制了数据集进行了实验. 最终结果表明, 本文的方法能够弥补现有方法的不足, 取得良好的反混淆效果, 为模型的反混淆方法提供了一种可行且有效的方案. 但是, 在本文的翻译过程中仍存在一些可以改进的地方. 首先, 本文只在小规模的数据集上验证了该方法的可行性, 而在更大的数据集及模型对各模块将会有更高的要求; 其次, 混淆信息提取模型对不同转换的混淆特征提取能力是不同的, 在本文的实现中, 只是将混淆信息提取为包含与不包含, 这使得仍然可能出现无法区分不同模型架构混淆到同一模型架构的情况, 因此也需要更细致的模型提取信息处理以进一步提高反混淆效果, 例如, 自动区分混淆转换属于上述 6 类转换中的哪一类等; 最后, 反混淆的技术能够为混淆方法提供一定的改进指导, 下一步可以进行混淆工具的进一步研究.

参考文献

1 Li JT, He ZZ, Rakin AS, *et al*. NeurObfuscator: A full-stack obfuscation tool to mitigate neural architecture stealing. International Symposium on Hardware Oriented Security and Trust (HOST). Tysons Corner: IEEE, 2021. 248–258. [doi: 10.1109/HOST49136.2021.9702279]

2 Oliynyk D, Mayer R, Rauber A. I know what you trained last summer: A survey on stealing machine learning models and defences. ACM Computing Surveys, 2023, 55(14s): 324. [doi: 10.1145/3595292]

3 Yuan XY, He P, Zhu QL, *et al*. Adversarial examples: Attacks and defenses for deep learning. IEEE Transactions on Neural Networks and Learning Systems, 2019, 30(9): 2805–2824. [doi: 10.1109/TNNLS.2018.2886017]

4 Hu X, Liang L, Li SC, *et al*. DeepSniffer: A DNN model extraction framework based on learning architectural hints. Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems. Lausanne: ACM, 2020. 385–399. [doi: 10.1145/3373376.3378460]

5 麦络, 董豪. 机器学习系统: 设计和实现. 北京: 清华大学出版社, 2023.

6 Stahlberg F. Neural machine translation: A review. Journal of Artificial Intelligence Research, 2020, 69: 343–418. [doi: 10.1613/jair.1.12007]

7 Ahmadi MM, Alrahis L, Colucci A, *et al*. NeuroUnlock: Unlocking the architecture of obfuscated deep neural networks. Proceedings of the 2022 International Joint Conference on Neural Networks (IJCNN). Padua: IEEE, 2022. 1–10. [doi: 10.1109/IJCNN55064.2022.9892545]

8 Chen TQ, Goodfellow IJ, Shlens J. Net2Net: Accelerating learning via knowledge transfer. Proceedings of the 4th International Conference on Learning Representations. San Juan, 2016.

9 Wistuba M. Deep learning architecture search by neuro-cell-based evolution with function-preserving mutations. Proceedings of the 2019 European Conference on Machine Learning and Knowledge Discovery in Databases. Dublin: Springer, 2019. 243–258. [doi: 10.1007/978-3-030-10928-

8_15]
10 Zhu H, An ZL, Yang CG, *et al*. EENA: Efficient evolution of neural architecture. Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision Workshops. Seoul: IEEE, 2019. 1891–1899.
11 Zhou T, Ren SL, Xu XL. ObfuNAS: A neural architecture search-based DNN obfuscation approach. Proceedings of the 41st IEEE/ACM International Conference on Computer-aided Design (ICCAD). San Diego: ACM, 2022. 81. [doi: 10.1145/3508352.3549429]
12 Duddu V, Samanta D, Rao DV, *et al*. Stealing neural networks via timing side channels. arXiv:1812.11720, 2018.
13 Hua WZ, Zhang ZR, Suh GE. Reverse engineering convolutional neural networks through side-channel information leaks. Proceedings of the 55th Annual Design Automation Conference. San Francisco: ACM, 2018. 4. [doi: 10.1145/3195970.3196105]
14 Batina L, Bhasin S, Jap D, *et al*. CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel. Proceedings of the 28th USENIX Conference on Security Symposium. Santa Clara: USENIX Association, 2019. 515–532.
15 Yu HG, Ma HC, Yang KC, *et al*. DeepEM: Deep neural networks model recovery through EM side-channel information leakage. Proceedings of the 2020 International Symposium on Hardware Oriented Security and Trust (HOST). San Jose: IEEE, 2020. 209–218. [doi: 10.1109/HOST45689.2020.9300274]
16 葛继科, 邱玉辉, 吴春明, 等. 遗传算法研究综述. 计算机应用研究, 2008, 25(10): 2911–2916. [doi: 10.3969/j.issn.1001-3695.2008.10.008]
17 Zhou MY, Gao X, Wu J, *et al*. ModelObfuscator: Obfuscating model information to protect deployed ML-based systems. Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis. Seattle: ACM, 2023. 1005–1017.
18 Xu H, Su YX, Zhao ZR, *et al*. DeepObfuscation: Securing the structure of convolutional neural networks via knowledge distillation. arXiv:1806.10313, 2018.
19 Klein G, Kim Y, Deng YT, *et al*. OpenNMT: Open-source toolkit for neural machine translation. Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics —System Demonstrations. Vancouver: ACL, 2017. 67–72.
20 Yang SH, Wang YX, Chu XW. A survey of deep learning techniques for neural machine translation. arXiv:2002.07526, 2020.
21 Gehring J, Auli M, Grangier D, *et al*. Convolutional sequence to sequence learning. Proceedings of the 34th International Conference on Machine Learning (ICML). Sydney: PMLR, 2017. 1243–1252.
22 Chen MX, Firat O, Bapna A, *et al*. The best of both worlds: Combining recent advances in neural machine translation. Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Vol. 1: Long Papers). Melbourne: ACL, 2018. 76–86.
23 Vaswani A, Shazeer N, Parmar N, *et al*. Attention is all you need. Proceedings of the 31st International Conference on Neural Information Processing Systems. Long Beach: Curran Associates Inc., 2017. 6000–6010.
24 Lattner C, Amini M, Bondhugula U, *et al*. MLIR: A compiler infrastructure for the end of Moore's law. arXiv:2002.11054, 2020.
25 Mikolov T, Chen K, Corrado G, *et al*. Efficient estimation of word representations in vector space. Proceedings of the 1st International Conference on Learning Representations. Scottsdale, 2013.
26 Freitag M, Al-Onaizan Y. Beam search strategies for neural machine translation. Proceedings of the 1st Workshop on Neural Machine Translation. Vancouver: ACL, 2017. 56–60.
27 He KM, Zhang XY, Ren SQ, *et al*. Deep residual learning for image recognition. Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Las Vegas: IEEE, 2016. 770–778.
28 Szegedy C, Liu W, Jia YQ, *et al*. Going deeper with convolutions. Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Boston: IEEE, 2015. 1–9.
29 Navarro G. A guided tour to approximate string matching. ACM Computing Surveys, 2001, 33(1): 31–88. [doi: 10.1145/375360.375365]

(校对责编: 张重毅)