Estudo e Avaliação de Plataformas Seguras de Deep Learning

João Barreira (a73831)
Rafael Costa (a61799)

Maio 2019

Universidade do Minho
Escola de Engenharia

Laboratório em Engenharia Informática
MEI / MIEI – Universidade do Minho, Braga, Portugal

Resumo

Atualmente, tem vindo a aumentar o número de aplicações de sistemas de Deep Learning nos mais variados contextos. No entanto, alguns destes contextos exigem que sejam tomadas precauções no que toca à privacidade e segurança dos dados utilizados. Neste contexto, várias frameworks têm surgido, procurando dar resposta a este problema utilizando técnicas como Differential Privacy, Federated Learning, Secure Multiparty Computing ou Homomorphic Encryption. Assim sendo, este trabalho visa fazer um estudo que permita comparar estas diferentes soluções do ponto de vista do utilizador comum, que possui recursos computacionais limitados. Para tal, selecionaram-se três destas frameworks (TensorFlow Privacy, PySyft e TF Encrypted), tendo em conta a viabilidade de um estudo aprofundado e, utilizando o dataset MNIST, efetuou-se um estudo comparativo tendo em conta vários parâmetros como o tempo de treino e accuracy, bem como a utilização dos recursos da máquina (tempo de CPU, RAM, etc.). Deste modo, o objetivo passou por medir o desempenho das frameworks quando aplicadas a situações concretas que exijam a privacidade dos dados. Foi concluído que a framework TensorFlow Privacy fornece um mecanismo de privacidade bastante rigoroso, mas requer um elevado número de recursos computacionais, não sendo recomendada para aplicações que necessitam de alto desempenho. Além disso, é verificado que a framework TF Encrypted apresenta os piores resultados em termos de accuracy possivelmente derivado de operações sobre dados encriptados. É também verificado que a framework PySyft poderá possuir overheads muito mais elevados quando aplicada num contexto real e que possui uma implementação menos rigorosa de privacidade de dados.

Conteúdo

1 Introdução

1.1 Contextualização

O termo Machine Learning (ML) pode ser descrito como um conjunto de decisões automatizadas e deteção de padrões relevantes num enorme volume de dados, apresentando uma diversidade de algoritmos que efetuam o tratamento e retenção de informações relevantes dos mesmos, de modo a poderem aplicar decisões de acordo com todo o processo de aprendizagem obtido. Nas últimas décadas, e com o aumento da quantidade de dados produzida por diversos tipos de dispositivos, surgiram inúmeras tecnologias baseadas em ML, tais como mecanismos de pesquisa e sistemas de recomendação que oferecem os melhores resultados para uma determinada pesquisa, ou informações relevantes baseadas nas decisões efetuadas por parte de outros utilizadores. Além destas aplicações, permite implementar mecanismos anti-spam de modo a filtrar mensagens indesejáveis no e-mail, assim como a criação de software de deteção de fraudes associadas a transações de cartões de crédito. Machine Learning também é vastamente utilizada em áreas como classificação de imagem, bioinformática, medicina e astronomia [1].

Um conceito aliado ao de ML é o de Deep Learning (DL). Deep Learning é um ramo de ML associado a modelos computacionais compostos por um conjunto de múltiplas camadas que permitem a aprendizagem de representações de dados a partir de vários níveis de abstração, compostas por unidades designadas de neurónios. Este conceito está associado às Redes Neuronais Artificiais, assemelhando-se ao processo de aprendizagem dos seres humanos. Este método de ML melhorou significativamente o estado da arte no que toca a reconhecimento de voz, reconhecimento e deteção de objetos visuais, assim como processamento de imagem, vídeo e áudio [2].

No âmbito dos conceitos descritos, surge uma vasta diversidade de plataformas criadas com o intuito de permitir o desenvolvimento de software que tire partido das vantagens associadas a esses mesmos conceitos, tais como o TensorFlow e PyTorch. Aliadas a essas plataformas, tem sido criada uma diversidade de frameworks ou extensões das mesmas com vista a proporcionar melhorias no que toca a abstrações de alto nível (e.g. Keras) ou outros assuntos de extrema relevância como a proteção de inúmeros registos que, de outro modo, possam comprometer a privacidade de um conjunto de utilizadores.

1.2 Motivação

Nos dias de hoje, tem vindo a aumentar consideravelmente a adoção de sistemas de ML nos mais variados ramos e indústrias. Estes sistemas são utilizados por forma a encontrar padrões relevantes numa enorme quantidade de dados disponível. Atualmente existe já um grande leque de aplicações destes sistemas, que resultam em contribuições positivas em várias áreas do conhecimento.

No entanto, em múltiplos contextos de aplicação destes sistemas, existe uma necessidade em garantir a privacidade dos dados utilizados. Alguns exem-

plos poderão ser o caso dos dados médicos utilizados para a prevenção de doenças, informação bancária para concessão de créditos e deteção de fraude, ou ainda dados provenientes do conteúdo de mensagens de correio eletrónico, por forma a detetar fontes de spam. Nos casos mencionados, existe, inclusivamente, uma responsabilidade legal na preservação da confidencialidade médica ou bancária dos utilizadores.

Assim sendo, existe, cada vez mais, uma necessidade em procurar balancear estes dois fenómenos. Por um lado, existe uma procura crescente no que toca à obtenção de uma grande quantidade de novos dados por parte dos utilizadores dos mais variados serviços. Por outro lado, impõe-se que sejam tomados cuidados no que diz respeito à privacidade dos utilizadores, por forma a ser respeitado o direito à confidencialidade em diversos enquadramentos.

Neste contexto, têm vindo a surgir várias frameworks que procuram dar resposta a este tipo de problemas, fazendo-o utilizando diferentes abordagens. Face à multitude de implementações existentes atualmente, existe uma necessidade de se efetuar um estudo aprofundado de comparação e de benchmarking.

1.3 Objetivos

Os principais objetivos deste projeto podem ser divididos em duas fases distintas. A primeira fase, consiste em estudar um conjunto de frameworks de DL a um nível teórico, identificando as diversas estratégias e algoritmos, assim como todos os workloads e testes efetuados às mesmas.

A segunda fase diz respeito a uma definição de um conjunto de métricas, workloads e ambientes de execução que representam a metodologia de testes a efetuar a cada uma das frameworks. O principal objetivo desta fase consiste em verificar como cada uma das frameworks se comporta fazendo variar a quantidade de recursos computacionais, efetuando-se uma comparação justa para os mesmos workloads e para os mesmos ambientes de execução.

Estudos como a verificação de privacidade dos dados, assim como uma comparação exaustiva de todos os parâmetros e respetiva classificação das frameworks saem fora do âmbito deste projeto. No entanto, este projeto pode servir de ponte para tais estudos mais aprofundados.

1.4 Apresentação do Caso de Estudo

Como caso de estudo, propõe-se um conjunto de seis frameworks de DL seguro, desenvolvidas nas plataformas TensorFlow e PyTorch, sendo que cada uma destas apresenta diferentes algoritmos de segurança. As frameworks propostas são a TensorFlow Privacy, coMind Federated Learning, TF Encrypted, nGraph-HE, SLALOM e PySyft, sendo que são estudadas as estratégias de privacidade de differential privacy, federated learning, secure multi-party computation, homomorphic encryption e uso de software seguro.

1.5 Estrutura do Relatório

A estrutura do relatório inicia-se pela secção 2, que começa com uma introdução aos conceitos teóricos que são utilizados pelas frameworks analisadas, como forma a garantir a segurança e privacidade dos dados. De seguida, analisa-se cada uma destas frameworks em concreto, tendo por base o artigo que lhes deu origem, realizado pelas equipas desenvolvedoras. Devido ao contexto da realização deste trabalho, houve necessidade de descartar algumas das frameworks estudas, tanto pelo facto de não corresponderem ao intuito do estudo (e.g. utilizarem compilares gráficos ou hardware específico), como simplesmente pelo facto de não ser possível realizar um estudo comparativo extensivo que albergasse um grande número de frameworks. Assim sendo, na subsecção seguinte é feito um estudo da viabilidade de cada uma das frameworks, e, consequentemente, feita a seleção das três que seriam alvo do estudo de comparação. Finalmente, tecem-se algumas considerações relativamente à não utilização de benchmarks previamente existentes.

A secção 3 compreende todas as observações relativas à metodologia adotada para a realização dos testes a cada uma das frameworks selecionadas. Começa-se por definir quais os objetivos dos testes, detalhando quais as métricas que seriam analisadas para o conseguir. De seguida, apresentam-se os ambientes de execução nos quais os testes irão decorrer, bem como quais os workloads que serão utilizados. A descrição dos workloads passa não só pela fundamentação da escolha dos datasets utilizados, mas também de quais os modelos referentes a cada uma das frameworks. Por fim, é feita uma descrição dos testes efetuados e scripts utilizados para a sua automatização.

Na secção 4 são apresentadas as tabelas que contêm os resultados dos testes efetuados. De seguida, é feito um levantamento de algumas observações relativas aos dados obtidos e feita a sua posterior análise, tendo como base os algoritmos e as características das frameworks estudados.

Finalmente, na secção 5, segue-se a conclusão e sugestões para um trabalho futuro.

2 Frameworks de Deep Learning Seguro

Nesta secção são descritos todos os algoritmos de segurança envolvidos no funcionamento das frameworks estudadas. Para cada framework são detalhados o seu modo de funcionamento, melhorias ou alternativas propostas aos algoritmos de proteção de dados, dando-se destaque aos testes efetuados por parte dos desenvolvedores, assim como as conclusões derivadas de tais testes. No final, é avaliada a viabilidade da seleção de cada framework para o nosso estudo, seguida de uma análise a sistemas de benchmark.

2.1 Algoritmos de Privacidade

2.1.1 Differential Privacy

Este tipo de algoritmo de segurança visa melhorar a proteção de dados face a outras técnicas de segurança como a data anonymization, cujo processo consiste em remover certos atributos que possam comprometer a privacidade de um determinado utilizador, tais como, por exemplo, o nome de uma pessoa ou a sua morada. O processo de data anonymization consegue, com sucesso, garantir a privacidade dos diferentes utilizadores, sendo suscetível, no entanto, a um tipo de ataques chamado linkage attacks. Neste tipo de ataques são cruzados os dados provenientes de vários datasets semelhantes mas de fontes distintas, de modo a se conseguir inferir os atributos protegidos nesses mesmos datasets. Um exemplo deste tipo de ataques ocorreu em 2007, numa competição proposta pela Netflix quando foi disponibilizado um dataset que continha dados relativos a user ratings, sem comprometer a identidade de todos os utilizadores envolvidos. No entanto, um conjunto de analistas conseguiu obter tais identidades com o auxílio de informações provenientes da IMDb. O mecanismo de differential privacy (DP) tem como objetivo prevenir este tipo de ataques ao introduzir ruído nas respostas, tornando o valor dessas mesmas respostas não confiável [3, 4].

2.1.2 Federated Learning

Os dispositivos móveis estão cada vez mais preparados para a recolha de dados por parte dos utilizadores de modo a melhorar a user experience. No entanto, todos estes dados podem comprometer a privacidade quando estes são transferidos para um servidor centralizado. Face a este tipo de questões surgiram implementações que permitem que os datasets de treino locais ao dispositivo não sejam transferidos para o servidor global. Em vez disso, cada cliente (i.e. dispositivo mobile) executa as computações necessárias localmente e apenas faz o upload dos updates necessários para o servidor global. O resultado final diz respeito a uma agregação de todos os updates efetuados localmente em cada cliente. Isto permite apenas a transferência de dados necessários para melhorar a qualidade do modelo global sem ter que se transferir todos os dados que poderiam comprometer a privacidade de um utilizador. Além disso, este tipo de técnica permite reduzir os riscos de privacidade e de segurança diminuindo a superfície suscetível a ataques, ou seja, apenas os dispositivos móveis podem sofrer ataques de segurança, ao invés dos dispositivos móveis em conjunto com o servidor global [6].

2.1.3 Secure Multiparty Computation

Secure Multiparty Computation (SMPC) é uma vertente de criptografia que aglomera um conjunto de técnicas que permitem que diferentes entidades avaliem operações ou funções, sem revelarem os seus inputs privados. Um exemplo deste tipo de processo pode ser descrito através de duas entidades A e B

que detêm, respetivamente, os inputs x e y e acordam mutuamente em executar a seguinte função:

$$F(x, y) = max(x, y)$$

Os inputs apenas são conhecidos pela entidade que o forneceu. Assim, as entidades descritas seguem um protocolo de modo a obterem o output desejado. Neste caso, as entidades aprendem através do output recebido do seus próprios inputs. Por exemplo, caso o resultado da função acima seja y, então a entidade B apenas obtém conhecimento que o seu valor submetido corresponde ao valor máximo. No entanto, esta entidade desconhece por completo qual é o valor x de A. Este cenário pode ser facilmente generalizado para entidades que possuem um conjunto diverso de inputs e outputs e para funções que produzem diferentes outputs para diferentes entidades [7, 8].

2.1.4 Homomorphic Encryption

Um dos desafios relacionados com o trabalho desempenhado pelas mais variadas ferramentas de ML atuais, consiste em executar as tarefas que possibilitem aos utilizadores retirarem algum tipo do conhecimento dos seus dados, protegendo as informações que estes contêm.

Uma das soluções para este problema é a Encriptação Homomórfica (HE), que faz com que seja possível efetuar computações sobre datasets cujos dados se encontram encriptados. Desta forma, os utilizadores podem encriptar os seus dados, enviá-los para um sistema computacional que não tem acesso à chave criptográfica – mas que consegue efetuar a computação graças à HE – e, finalmente, receber o resultado da computação de forma encriptada, que poderá, só então, ser analisado.

No entanto, este tipo de sistemas têm algumas limitações conhecidas:

- **Funções suportadas:** Alguns sistemas apenas suportam uma operação algébrica (adição ou multiplicação), sendo designados de "parcialmente homomórficos" (PHE). Outros suportam ambas estas operações (multiplicação e adição) e são denominados "completamente homomórficos" (FHE).
- **Computational depth:** O ruído estatístico introduzido por estes sistemas tende a acumular-se, levando a que seja impossível fazer a descriptografia dos dados encriptados.
- **Campos de dados numéricos:** A maior parte destes sistemas operam com números inteiros ([18, 15]), enquanto outros com booleanos ([19]) ou números reais ([16]). Um dos problemas em sistemas que utilizam inteiros encontra-se em escalar a magnitude dos números por um fator menor que 1.
- **Custo computacional:** Os sistemas que utilizam HE consomem tipicamente recursos computacionais (e.g. tempo de CPU, memória) numa

ordem de grandeza várias vezes maior do que os sistemas equivalentes que operam sobre dados não encriptados.

2.2 Frameworks Estudadas

2.2.1 Google's TensorFlow Privacy

Esta framework open source desenvolvida pela Google foi criada com o intuito de facilitar aos desenvolvedores a proteção dos dados provenientes de um universo de utilizadores, sendo aplicada a uma das ferramentas mais populares de ML, o TensorFlow. De modo a garantir a privacidade dos dados, esta framework usa uma técnica denominada de Private Aggregation of Teacher Ensembles (PATE), em que é efetuada uma agregação com ruído das respostas provenientes de um conjunto de teachers que operam sobre dados disjuntos. Estas respostas são transferidas para um modelo chamado student. Devido à introdução de ruído nas respostas, pode-se dizer que esta técnica assenta numa metodologia de privacidade DP, descrita na secção 2.1.1. Este mecanismo de privacidade encontra-se dividido nos seguintes componentes [4]:

Teacher models. Cada teacher corresponde a um modelo que opera sobre um subconjunto independente dos dados que pretende proteger, de modo a garantir-se que nenhum teacher opere sobre porções de dados já tratadas por outro. Pode ser aplicada qualquer técnica de aprendizagem a cada teacher, resultando num conjunto de n modelos distintos que resolvem a mesma tarefa. O objetivo destes modelos consiste em prever um conjunto de labels através de inferência.

Mecanismo de agregação. Quando existe um consenso muito forte entre os diferentes teachers, pode-se afirmar que a classificação efetuada pelos mesmos não depende do modelo de aprendizagem. Assim sendo, esta decisão é realizada de um modo privado e não depende de um determinado ponto específico do conjunto total de dados de treino, visto esse ponto específico apenas poder estar presente no subconjunto de treino de apenas um dos diferentes teachers. De modo a se garantir uma DP rigorosa, o mecanismo de agregação da técnica de PATE original consiste numa contagem dos votos atribuídos a cada classe, adicionando ruído laplaciano a esses mesmos votos. O output consiste na classe com o maior número de votos com ruído. Este tipo de mecanismo é chamado de max-of-Laplacian ou LNMax.

Student model. A restante componente da técnica de PATE consiste no treino do student a partir das respostas provenientes do conjunto de teachers através do acesso de dados públicos mas do tipo unlabeled. De modo a se poder diminuir o custo associado à privacidade no processo de categorização dos dados, apenas é permitido um número fixo de queries de modo a se efetuar o treino do student. Esta decisão é importante já que o processo de previsão pelo conjunto de teachers aumenta o custo associado à privacidade e seria inviável para um número ilimitado de queries. Para além disso, o número fixo de queries diminui o número de ataques em que se efetua uma análise do modelo de modo a recuperar os dados de treino.

A framework desenvolvida não segue a abordagem original do mecanismo de PATE, já que usa a introdução de ruído gaussiano nas respostas produzidas pelos teachers ao invés de ruído laplaciano. Esta decisão permitiu uma diminuição na quantidade de ruído necessária para se atingir a privacidade das queries efetuadas pelo student face à implementação original. Outra questão avaliada consistiu no consenso entre os diferentes teachers. Quando ocorre uma grande discordância entre os teachers, é muito provável que o agregador faça o output de uma classificação errada a uma determinada query. Os desenvolvedores desta framework verificaram que quando isto acontece, ocorre um elevado custo em termos de privacidade. Além disso, estes tipos de resultados são bastante maus no processo de treino do modelo do student. Assim, o agregador desenvolvido apenas responde a queries quando ocorre um grande consenso entre os diversos teachers.

O tipo de agregador desenvolvido, exclui as queries mais custosas em termos de privacidade, mas ignora a possibilidade de o modelo do student receber classificações que contribuem muito pouco para a qualidade de aprendizagem. De modo a reverter esta situação, foi também incorporado um agregador interativo que descarta queries às quais o modelo do student é capaz de produzir as mesmas classificações por parte dos teachers.

Workloads e testes efetuados. Foram efetuados testes para um conjunto de diferentes métricas tais como a accuracy para diferentes datasets. Além disso foram efetuadas comparações entre soluções anteriores e a solução proposta por esta framework. Os testes efetuados foram os seguintes:

- **MNIST, SVHN, e UCI Adult databases:** O objetivo dos testes efetuados com estes datasets foi o de avaliar o quanto o agregador desenvolvido à custa de ruído gaussiano seria eficaz face à antiga proposta de PATE original baseada em LNMax. Para os datasets MNIST e SVHN foram usadas redes convolucionais para os modelos dos teachers, treinadas em diferentes partições dos datasets com recurso a uma aprendizagem semi-supervisionada. Já para o UCI Adult, cada teacher correspondeu a uma random forest com um tipo de aprendizagem puramente supervisionada. Os dados de teste foram partidos em duas metades, sendo que a primeira metade foi usada para os inputs não classificados de modo a simular os dados públicos provenientes do modelo do student. Já a segunda metade foi usada para avaliar a performance dos testes efetuados. Estes testes foram efetuados para uma diversidade de parâmetros intrínsecos à framework, como por exemplo, o número de teachers. Os resultados foram comparados com os resultados reportados por Papernot et al. (2017) [5].

- **Glyph:** O objetivo deste teste foi o de verificar como a framework se comportava em termos de custos de privacidade face às antigas aplicações de PATE. Para isso foram criados modelos à custa do dataset Glyph, que consistia no dataset com o maior número de classes avaliado pelas propostas de PATE até ao momento. Numa fase inicial deste teste, houve um

pré-processamento do dataset, em que se removeram classes com poucas amostras ou demasiado ambíguas para uma correta classificação. Cada teacher recebeu uma porção dos dados de treino e consistiu numa ResNet construída à custa de 32 camadas leaky ReLU. O treino foi efetuado com batches de 100 inputs de 40K steps através de gradiente descendente estocástico com momentum. Quanto ao learning rate, este foi inicializado com o valor 0.1, diminuindo ao longo de 10K steps para 0.01 e no final de 20k steps para 0.001.

A arquitetura do student model seguiu uma rede convolucional com um modelo de aprendizagem semi-supervisionado com virtual adversarial training. O treino foi efetuado com otimizador Adam por 400 épocas e com uma taxa de aprendizagem igual a $6e10^{-5}$.

Através dos testes descritos acima, provou-se uma melhoria do algoritmo PATE pela introdução de ruído gaussiano e pelas modificações efetuadas ao agregador das respostas provenientes dos teacher models.

2.2.2 CoMind Federated Learning

Desenvolvida com recurso à plataforma TensorFlow, esta framework está baseada no algoritmo de federated learning (FL) descrito na secção 2.1.2 e visa assegurar mecanismos de privacidade em DL para dispositivos móveis. Neste contexto, os desenvolvedores desta ferramenta tiraram como ponto de referência o algoritmo de FL original, tentando otimizar certas questões relacionadas com o mesmo. Os conceitos tidos em conta foram os seguintes:

- **Non independent and identically distributed (Non-IID):** O conjunto de dados de treino está diretamente relacionado com a utilização de um certo dispositivo móvel por parte de um cliente. Assim sendo, um determinado dataset local proveniente de um utilizador não pode ser usado para representar toda a população.
- **Unbalanced:** Indo de encontro ao primeiro conceito, o uso de dispositivos móveis varia imenso de utilizador para utilizador, o que resulta em conjuntos de dados de treino com dimensões bastante distintas.
- **Limited communication:** Os dispositivos móveis encontram-se muito frequentemente em modo offline ou sujeitos a comunicações muito lentas e dispendiosas.

Em problemas de otimização de FL, os custos relacionados com a comunicação com o servidor global ultrapassam os custos computacionais. Tipicamente, é esperada uma largura de banda de 1 MB/s, ou menos, para os uploads para o servidor e, para além disso, é esperado que os clientes apenas participem com updates algumas vezes ao dia, quando ligados a uma rede wi-fi. Por outro lado, a computação realizada apresenta um baixo custo já que o dataset mantido localmente em cada dispositivo é bastante inferior ao dataset total,

aliado ao facto de os dispositivos móveis modernos apresentarem bons recursos computacionais. Assim, o objetivo proposto por esta framework consistiu em aumentar o nível de computação local aos dispositivos, diminuindo o número de comunicações necessárias para se efetuar o treino do modelo. Foram estudadas duas alternativas possíveis: aumentar o paralelismo, onde eram usados mais clientes que efetuavam processos independentes a cada comunicação, ou aumentar a computação em cada cliente, onde cada cliente efetuava cálculos bastante complexos ou invés de operações simples como o cálculo de gradientes [6].

A maioria das otimizações efetuadas em DL baseiam-se no gradiente estocástico descendente (SGD), pelo que foi considerado como exemplo base para as otimizações realizadas pelos desenvolvedores desta framework. Partiu-se do princípio que era apenas efetuado o cálculo do gradiente num único batch por comunicação entre um determinado cliente e o servidor global. Este processo é eficiente, mas requer um grande número de rondas de treino para a produção de bons modelos. Assim, a estratégia inicial passou por selecionar uma fração de clientes a cada ronda e calcular o gradiente a partir de todos os dados provenientes desses mesmos clientes. Foi estipulado o uso de batch sizes com elevados valores, sendo estes correspondentes à fração de clientes existente. Ou seja, para apenas um cliente este algoritmo passaria a ser gradiente completo descendente (não estocástico). Os desenvolvedores denominaram esta estratégia FederatedSGD ou FedSGD.

Outra variante do processo descrito caracteriza-se pelo cálculo de apenas um passo do gradiente descendente por parte do cliente apenas com os dados locais. Depois disso, é calculada a média dos pesos de todos os modelos resultantes por parte dos clientes envolvidos. Através deste processo, pode ser adicionada mais computação a cada cliente ao iterar, por um determinado número de vezes, os updates locais antes do cálculo da média. Esta variante foi chamada FederatedAveraging. A computação é controlada por três parâmetros: a fração de clientes, o número de épocas realizadas aos datasets locais a cada ronda de comunicação com o servidor e o tamanho do minibatch. Assim, um tamanho de minibatch igual ao tamanho do dataset local e uma época, correspondem exatamente à alternativa FedSGD.

Workloads e testes efetuados. Foram efetuados testes para modelos relacionados com classificação de imagem e de processamento de texto. Foi inicialmente escolhido um modelo com uma dimensão suficiente de modo a avaliar a hiperparametrização do algoritmo FederatedAveraging. Para testar quão eficaz é este algoritmo, os desenvolvedores criaram testes que avaliaram modelos de linguagem de grandes dimensões.

- **MNIST:** Foram cridos dois modelos distintos em torno deste dataset. O primeiro modelo consistiu numa multilayer-perceptron com duas hidden layers com 200 unidades que continham como função de ativação a função ReLU. O segundo modelo foi uma rede neuronal convolucional com duas 5x5 camadas convolucionais (a primeira com 32 canais, a segunda com 64

e as seguintes com 2x2 max pooling), uma camada completamente ligada constituída por 512 unidades e como função de ativação a função ReLU. A camada de output continha a função de ativação softmax.

Foram estudados dois métodos de partição do dataset MNIST pelos vários clientes envolvidos no processo de FL: o método IID, onde os dados foram baralhados e divididos entre 100 clientes, sendo que cada cliente recebeu 600 exemplos; e o método Non-IID, onde foi efetuada uma ordenação dos dados por dígito, sendo posteriormente divididos em 200 partes com 300 exemplos. No último caso, foram também criados 100 clientes, sendo que cada cliente recebeu duas partes dos dados.

- **Dataset construído a partir da obra *The Complete Works of William Shakespeare*:** Foi criado um dataset a partir desta obra literária para se efetuarem testes relativos a processamento de texto. Foram construídos datasets ao nível dos clientes para cada papel de cada peça da obra com pelo menos duas linhas de texto. O resultado consistiu num dataset com 1146 clientes, sendo que para cada cliente foram selecionados 80% dos dados para treino e 20% dos dados para teste. Além desta versão que se traduzia como um dataset bastante desequilibrado em termos de dados (as peças possuíam diferentes números de linhas de texto), foi também criada uma versão balanceada e IID também com 1146 clientes.

  Os dados foram submetidos a uma LSTM de processamento de texto, sendo que foi efetuada uma previsão do próximo caratere após o processamento de todos os carateres de uma linha de texto. Foram sobrepostas duas camadas de LSTMs, cada uma com 256 nós. O resultado da segunda camada de LSTM é enviado para uma camada de output com softmax com um nó por caratere. O tamanho da janela temporal da LSTM considerado foi de 80 carateres.

- **CIFAR-10:** Foram efetuadas experiências com este dataset de modo a verificar a eficácia do algoritmo FederatedAveraging. O dataset foi dividido entre 100 clientes, sendo que cada um continha 500 exemplos de treino e 100 exemplos de teste. A arquitetura do modelo desenvolvido seguiu o tutorial da plataforma TensorFlow que consiste em duas camadas convolucionais seguidas por duas camadas completamente ligadas e uma camada linear para produzir logits.

Os testes mencionados acima demonstraram a capacidade de uso de FL, em particular a variante FederatedAveraging, para problemas de privacidade em DL. No entanto, é descrito que o uso de algoritmos de segurança como DP (secção 2.1.1) ou SMPC (secção 2.1.3), em conjunto com a implementação proposta, podem garantir uma maior proteção de dados.

2.2.3 TF Encrypted

O intuito desta framework desenvolvida com recurso à plataforma TensorFlow foi o de tornar o processo de privacidade dos dados em ML muito

mais acessível. No processo de produção desta framework, os desenvolvedores focaram-se em quatro conceitos essenciais [9]:

- **Usabilidade:** Construção de uma plataforma familiar e intuitiva através de um aproveitamento das componentes presentes na plataforma TensorFlow.
- **Integração:** Redução das computações relacionadas com a proteção de dados a grafos do TensorFlow, o que permite uma mistura destas operações com outras de qualquer tipo.
- **Extensibilidade:** Ao tirar-se partido de abstrações de alto nível provenientes do TensorFlow, torna-se possível a implementação de novos protocolos de segurança tendo como base um conjunto de primitivas otimizadas. Além disso, mantém uma melhor legibilidade de código.
- **Performance:** Atinge-se um grande nível de eficiência através de engines de execução distribuída do TensorFlow otimizadas para networking, paralelismo e escalabilidade.
- **Benchmarking:** Ao combinar todas as componentes mencionadas acima, obtem-se uma framework comparável de ML privada.

A privacidade dos dados é garantida através de SMPC, descrito na secção 2.1.3, com recurso a uma variante de um protocolo chamado SPDZ. A ideia base deste protocolo consiste em trabalhar com dados aleatórios ao invés de encriptar os inputs provenientes das diversas entidades envolvidas. No final do protocolo são efetuadas algumas verificações e, no caso de algum comportamento malicioso ser detetado, é abortado todo este processo. A computação efetuada é dividida em duas fases, sendo que a primeira corresponde a um pré-processamento em modo offline e a segunda opera em modo online. Na segunda fase, são usados algoritmos para processar os resultados provenientes dos inputs das diferentes entidades com a aleatoriedade correlacionada produzida pela fase offline [10].

A aleatoriedade correlacionada consiste na multiplicação de um conjunto de triplos aleatórios partilhados secretamente. Como exemplo, considere-se o triplo (a, b, ab) para um valor aleatório a e um valor aleatório b. As entidades efetuam uma encriptação de a e b a partir de uma chave pública global, usam propriedades homomórficas para efetuar operações de somatórios e de multiplicações, sendo posteriormente efetuados protocolos de desencriptação distribuída de modo a receberem o resultado de ab.

Na nova variante do protocolo, proposta para a framework em questão, os desenvolvedores basearam-se em masked tensors para representar estados intermediários que proporcionam operações como multiplicações SPDZ, permitindo a generalização de triplos de modo a reduzir o esforço computacional.

Os resultados das operações efetuadas de DL nesta framework, são traduzidos para valores inteiros ao invés de valores de vírgula flutuante, sendo fornecidas duas hipóteses de expressar esses valores inteiros. A primeira consiste num

valor fixo $int64$ que fornece uma melhor performance e a segunda caracteriza-se como sendo um valor CRT-based $int100$ que oferece uma precisão mais rigorosa.

Workloads e testes efetuados. Foi efetuada uma benchmark de modo a estudar o protocolo SPDZ descrito acima, através do dataset MNIST. Como ambiente de execução, foram efetuados testes na Google Cloud Platform na região us-east1 com recurso a 36 vCPUs/60 GB de memória. O dataset foi testado para diferentes modelos de redes neuronais detalhados na tabela 1. Estes modelos foram verificados para os dois tipos inteiros propostos ($int64$ e $int100$) e para diferentes valores de batch size. Além disso, foram comparados com modelos com recurso ao TensorFlow sem qualquer operações de privacidade de dados.

Tabela 1: Arquiteturas das redes neuronais de teste. As camadas convolucionais são ilustradas pelos parâmetros (field size, channels, stride, padding) e as camadas AvgPool pelo tamanho da janela

Network A	Network B	Network C
FC (784, 128)	Conv (5, 16, 1, 1)	Conv (5, 20, 1, 1)
BatchNorm	BatchNorm	BatchNorm
ReLU (approx)	ReLU (approx)	ReLU (approx)
FC (128, 128)	AvgPool (2)	AvgPool (2)
BatchNorm	Conv (5, 16, 1, 1)	Conv (5, 50, 1, 1)
ReLU (approx)	BatchNorm	BatchNorm
FC (128, 10)	ReLU (approx)	ReLU (approx)
	AvgPool (2)	AvgPool (2)
	FC (256, 100)	FC (800, 500)
	BatchNorm	BatchNorm
	ReLU (approx)	ReLU (approx)
	FC (100, 10)	FC (500, 10)

Como resultados obtidos dos testes efetuados, verificou-se que foi atingida uma accuracy semelhante aos modelos do TensorFlow. Além disso, a accuracy atingida pelos diversos modelos foi bastante semelhante para os dois tipos $int64$ e $int100$, com indicação de que o último tipo poderá resultar numa melhor distribuição dos resultados de output segundo a métrica de divergência Kullback-leibler.

2.2.4 nGraph-HE

O processo de utilização de HE (secção 2.1.4) em conjunção com as frameworks existentes de ML (e.g. Tensorflow, MXNet, PyTorch) não se apresenta, atualmente, como uma tarefa simples. Isto deve-se ao facto de não existir uma

ferramenta que permita construir modelos que utilizem HE por forma a construir sistemas de ML em larga escala que garantam a privacidade dos dados.

Assim sendo, o objetivo desta framework passa por simplificar o processo de utilização de HE, através da criação de uma ferramenta que atua como um backend de HE para o compilador nGraph (Cyphers et al., 2018 [14]) e que permite aos utilizadores fazer o deployment dos seus modelos em hardware à sua escolha, tirando partido de sistemas criptográficos – HEAAN (Cheon et al., 2017 [16]) ou SEAL (Laine, 2018 [15]) –, por forma a conseguir operar sobre informação encriptada.

O nGraph é um exemplo de um compilador gráfico que tem vindo a ser utilizado por forma a facilitar a otimização dos modelos desenvolvidos numa qualquer framework de DL em hardware específico onde estes serão utilizados. Estes compiladores fazem-no compilando a framework de alto nível numa representação intermédia (um grafo computacional), fazendo com que não seja necessário gerar código de baixo nível para cada combinação diferente de framework e hardware. Os desenvolvedores do nGraph-HE tiraram então partido destas vantagens de modo a representar computações de DL utilizando HE, mantendo uma separação clara entre um e outro tipo de tecnologias.

Relativamente à implementação em concreto, este sistema possui dois componentes principais. Em primeiro lugar, tem-se o Storage Model que contém os parâmetros estáticos do esquema de encriptação: cryptographic context (polynomial degree, plaintext modulus, security parameter), que armazena as propriedades do criptossistema utilizado; e payload representation, que diz respeito aos componentes necessários para os processos de encriptação e desencriptação (informação a ser encriptada, encoding, plaintext, informação encriptada, etc.). Isto permite obter uma abstração entre o nGraph-HE e o criptossistema utilizado, facilitando o processo de adicionar um outro sistema que atualmente não seja suportado oficialmente. O segundo componente é a Assembly Language que define a codificação de operações de baixo nível (adição, subtração, multiplicação, raiz quadrada e negação) em termos de operações do nGraph.

Foram ainda realizadas algumas técnicas de otimização como operação entre plaintext e cyphertext, SIMD packing e extensiva paralelização OpenMP de várias operações (encriptação, desencriptação, multiplicação de matrizes (GEMM), convolução, etc.).

Workloads e testes efetuados. Por forma a testar o nGraph-HE, foram realizados dois tipos de testes: multiplicação de matrizes e aplicação a modelos de redes neuronais.

No primeiro conjunto de testes, procurou-se observar e determinar o impacto que tem a encriptação dos dados com vários níveis de segurança criptográfica nas operações de multiplicação de matrizes (GEMM). Para tal foi considerada a operação $AB + C$ para dois níveis de segurança criptográfica (80 e 256 bits), onde A, B e C são matrizes quadradas, e onde os dados de A são encriptados e os de B e C são em plaintext. Assim, pôde-se observar um claro overhead que contribuiu, por exemplo, para uma execução cerca de duas

vezes mais lenta para 256 bits face a 80, em matrizes quadradas de dimensão 45. No entanto, para matrizes de dimensão inferior (até 25) este overhead é significativamente menos acentuado.

Relativamente ao estudo da aplicação do nGraph-HE a modelos de redes neuronais, foi implementada a rede CryptoNets em TensorFlow e comparado o seu resultado com uma rede desenvolvida diretamente sobre o nGraph-HE com recurso à linguagem C++. Assim, foi possível observar que o overhead associado à integração do TensorFlow com o nGraph-HE (incluindo a compilação para o modelo intermédio do grafo computacional) é bastante reduzido quando comparado com a solução direta, correspondendo apenas a 0.5% do tempo total de execução. Complementarmente, foi também tirado partido da otimização SIMD packing, enunciada anteriormente, e registados os tempos de execução para dois níveis de segurança criptográfica.

2.2.5 SLALOM

Trusted Execution Environments (TEE) são ambientes de execução existentes no mercado (e.g. Intel SGX, ARM TrustZone, Sanctum) que utilizam mecanismos de segurança tanto ao nível do hardware como do software por forma a isolar código e dados sensíveis de ambientes não confiáveis (tirando partido da sua melhor performance) e assegurando a correção dos resultados recebidos.

No entanto, este isolamento é também responsável por um overhead considerável, não só devido ao custo computacional destas proteções propriamente ditas mas, por exemplo, no caso do Intel SGX, o facto de estar limitado a CPUs de computadores desktop, cuja performance é bastante inferior às GPUs ou a CPUs de servidores.

Assim sendo, SLALOM é uma framework cujo objetivo é minimizar este problema de performance, respondendo à pergunta lançada por Stoica et al., 2017 [20]: "How do we efficiently leverage TEEs for secure machine learning computations?". Mais concretamente, fá-lo no contexto das Deep Neural Networks (DNN), cuja execução é parcialmente terceirizada (i.e. outsourced) de um nó computacional confiável para outro nó computacional não confiável mas com um melhor desempenho. Fá-lo sem comprometer a segurança dos dados e em contraposição com outras soluções (Ohrimenko et al. 2016 [21], Hunt et al. 2018 [22], Cheng et al. 2018 [23]) que realizam todas as computações dentro do ambiente de execução confiável.

Para tal, tiram proveito do algoritmo de Freivald ([24]) por forma a verificar o resultado das operações de multiplicação de matrizes que são, então, terceirizadas para os nós computacionais mais rápidos. Este processo é adaptado para que seja possível enviar as operações lineares (i.e. convoluções, convoluções separáveis e dense layers) de redes neuronais para esses nós computacionais, enquanto que as computações não lineares (e.g. activações, pooling, etc.) são realizadas localmente no ambiente confiável. No entanto, estas últimas operações representam uma pequena fração do tempo total de execução, dominado pelas operações lineares.

Workloads e testes efetuados. De modo a testar a exequibilidade e a performance deste processo, foram realizados vários testes utilizando como nó confiável um processador Intel Core i7-6700 Skylake Quad-Core 3.40GHz como suporte para o TEE Intel SGX num sistema com 8GB de RAM. Como nó não confiável foi utilizada uma placa gráfica Nvidia TITAN XP.

Em relação às workloads, foram utilizadas, em primeiro lugar, benchmarks sintéticas para testar o tempo de execução do CPU em realizar as computações lineares (i.e. multiplicação de matrizes e convoluções) em comparação com o tempo necessário para apenas fazer a verificação do resultado destas operações (que são, neste caso, terceirizadas) com recurso ao algoritmo de Freivald. Neste primeiro conjunto de testes, pôde-se observar que terceirizar a multiplicação de matrizes quadradas seria 4 a 8 vezes mais rápido do que efetuar o cálculo localmente. No entanto, para matrizes maiores, limitações físicas ao nível da DRAM do Intel SGX fazem com que se incorra numa perda de performance tanto na computação como na verificação. Para o caso das convoluções, também foram registadas melhorias significativas ao terceirizar estas operações.

Em segundo lugar, foram utilizados dois modelos (VGG16 [25] e MobileNet [26]) por forma a avaliar o débito do modelo (i.e. número de forward passes por segundo) como métrica principal a ter em conta. O primeiro modelo é descrito como um modelo com boa performance, contrariamente ao segundo que é encarado como o pior cenário de teste para a aplicação do SLALOM. No caso do primeiro, foram registadas melhorias de 19.6 vezes ao nível do débito do modelo (com pré-processamento). Relativamente ao segundo modelo, pôde-se observar melhorias de 3.5 a 6.3 vezes também em relação ao débito.

2.2.6 OpenMined's PySyft

PySyft é uma framework criada pela OpenMined que tem como objetivo disponibilizar um conjunto de funcionalidades como MPC, FL e DP (secções 2.1.3, 2.1.2 e 2.1.1, respetivamente) no ambiente do PyTorch.

Para tal, o seu trabalho consistiu, primeiramente, na construção de um protocolo standard para a comunicação entre workers e que permite a existência do algoritmo de FL. De seguida, a criação de um modelo abstrato de tensors em cadeia que permite vários tipos de operações como, por exemplo, permitir a um worker enviar ou receber um tensor. Por fim, a disponibilização dos elementos necessários para a implementação dos algoritmos de DP e MPC utilizando esta nova framework.

Estrutura de tensors em cadeia. A execução de operações ou o envio de tensors para workers pode ser representada como uma sucessão de operações, sendo cada operação denotada, neste caso, por uma classe específica. Para tal, foi criada a abstração SyftTensor que representa estes estados e transformações da informação. Estes SyftTensors podem, então, ser encadeados na estrutura referida anteriormente que tem sempre em primeiro lugar um tensor do PyTorch, seguido de instâncias de SyftTensor representativas do seu papel, como o LocalTensor ou o PointerTensor. O primeiro é criado automaticamente quando um PyTorch tensor é instanciado e é responsável pela execução das operações

nativas (ao PyTorch) correspondentes. Já o segundo é criado sempre que um tensor é enviado para um worker remoto.

FL virtual e real. Por forma a simplificar o processo de debugging, esta framework implementou o conceito de Virtual Workers que existem dentro de uma mesma máquina, não comunicando através da rede. Estes apenas se encarregam de replicar o conjunto de operações e disponibilizar a mesma interface que os verdadeiros workers utilizam para comunicar entre si. Paralelamente, os workers que trabalham efetivamente em rede possuem duas implementações. A primeira utiliza sockets de rede standard sem qualquer outra funcionalidade. Os segundos tiram partido de web sockets, permitindo que vários workers sejam instanciados através de um browser.

Aplicação de MPC. Através da utilização dos PointerTensors descritos anteriormente, passa a ser possível a criação do MPCTensor, responsável pela aplicação do algoritmo de MPC. O dono do modelo é, então, designado de local worker, sendo os restantes designados de remote workers. No entanto, um dos problemas que ainda não foram resolvidos pelos autores desta framework prende-se no facto de não existir um mecanismo que certifique que cada um dos participantes atue de forma segura e bem intencionada.

Aplicação de DP. Foi também implementado um algoritmo de DP que teve por base o trabalho de Abadi et al (2016) [27]. Concretamente, foi implementado um Stochastic Gradient Descent (SGD), utilizando diretamente o privacy accountant proposto por Abadi et al. (2016) [27] mas implementando um sanitizer próprio que utiliza ruído gaussiano. Segundo a equipa de desenvolvimento, encontra-se atualmente em desenvolvimento uma implementação diferente de DL baseado no trabalho descrito em Papernot et al. (2017) [5], que, por sua vez, tem por base o modelo de student-teacher, falado anteriormente neste relatório na secção 2.2.1, correspondente à framework Google's TensorFlow Privacy.

Workloads e testes efetuados. O primeiro conjunto de testes realizado teve por base o Boston Housing dataset que corresponde a um conjunto de dados referentes a indicadores socioeconómicos dos subúrbios da cidade cidade de Boston. Neste teste pôde-se verificar que existe um overhead relativamente pequeno no que diz respeito à utilização dos Web Socket workers, em vez dos Virtual Workers normais, falados anteriormente. No entanto, quando comparado com o modelo de PyTorch nativo, a execução é 46 vezes mais lenta. Retiraram-se as mesmas conclusões relativamente ao overhead reduzido na utilização de Web Socket workers num segundo conjunto de testes utilizando o Pima Indian Diabetes dataset, referentes a dados de uma população indígena norte-americana.

Paralelamente, foi possível observar a degradação na accuracy e MSE à medida que se aumentava o nível de privacidade dos modelos de DP e face ao modelo base sem qualquer tipo de privacidade. Registou-se uma diferença de cerca de 9.7% (ao nível da accuracy) e 5.7 (relativamente ao MSE) face ao modelo base.

Face a estes resultados, como trabalho futuro, a equipa de desenvolvimento indica que a tarefa fundamental será reduzir o grande overhead encontrado no que diz respeito ao tempo de treino.

2.3 Viabilidade do Estudo das Frameworks

Antes de se efetuar qualquer estudo mais aprofundado às frameworks propostas, e tendo como base os conceitos detalhados nas secções anteriores, verificou-se a viabilidade de estudo a cada framework de acordo com os recursos disponibilizados. A tabela 2 ilustra um resumo das características das frameworks estudas, indicando, para cada uma destas, os algoritmos de segurança e datasets usados. É também indicado se uma determinada framework requer o uso de hardware específico para uma execução apropriada.

Tabela 2: Características das frameworks estudadas

Framework	**Algoritmo**	**Hardware específico?**	**Datasets**
Google's Tensorflow Privacy	PATE	Não	MNIST, SVHN, UCI e Glyph
CoMind Federated Learning	*Federated Learning*	Não	MNIST, CIFAR-10, Complete Works of William Shakespeare
Morten Dahl's tf-encrypted	SMPC	Não	MNIST
nGraph-HE	*Homomorphic Encryption*	Não	MNIST (pre-trained *CryptoNets*)
SLALOM	*Outsourcing* de operações lineares	Sim	ImageNet (pre-trained VGG16 and MobileNets)
OpenMined's PySyft	*Federated Learning*	Não	Boston Housing e Pima Indian Diabetes

Tal como foi indicado na secção 2.2.5, a framework SLALOM requer o uso de hardware seguro para um bom funcionamento. Face a este motivo, optou-se por descartar esta framework de um estudo mais aprofundado.

A framework nGraph-HE (secção 2.2.4) tem como principal objetivo mostrar a exequibilidade de operações de HE em hardware próprio, otimizado com recurso ao software Intel® nGraph Compiler. Na documentação presente no repositório `https://github.com/NervanaSystems/he-transformer` é descrito que esta framework não deve ser usada como um produto, mas sim como uma ferramenta de pesquisa. Devido ao contexto deste projeto, considera-se mais útil o estudo de potenciais frameworks a serem usadas no mercado. Além disso, a utilização de compiladores gráficos para otimização de modelos em hardware

especifico sai fora do âmbito do projeto. Devido a estas razões, abandonou-se a hipótese de um estudo intensivo desta framework.

Das restantes frameworks, verificou-se que duas delas usavam como algoritmo de segurança o processo de FL (secção 2.1.2). Neste trabalho de pesquisa tem mais valor o estudo e comparação de diversos algoritmos de segurança das frameworks do que propriamente a comparação de diversas implementações do mesmo algoritmo. Por este motivo, optou-se por retirar do estudo intensivo uma das frameworks de FL.

Assim sendo, foram escolhidas as frameworks TensorFlow Privacy, TF Encrypted e PySyft para se efetuar um estudo de comparação e de benchmarking.

2.4 Benchmarks

Para a criação dos vários testes a realizar, considerou-se o uso de algumas benchmarks, tais como a mlbench, Deep Learning Benchmarking Suite da Hewlett Packard, MLPerf, Deep Learning Benchmark da AWSLabs, assim como uma benchmark presente no repositório `https://github.com/guolinke/deep-learning-benchmarks`. No entanto, verificou-se que o uso de tais benchmarks era bastante complexo e difícil de adaptar para um conjunto de métricas que se pretendia avaliar. De modo a evitar uma curva de aprendizagem elevada e aliado ao facto de as frameworks escolhidas fornecerem tutoriais bastante detalhados sobre os seus modos de funcionamento, optou-se por não utilizar tais benchmarks para a criação dos testes de comparação.

3 Metodologia dos Testes Efetuados

Para se efetuar um estudo de comparação entre as frameworks escolhidas (secção 2.3), estipulou-se um conjunto de métricas a avaliar, definiram-se ambientes de execução e workloads apropriados e, finalmente, desenvolveram-se scripts para automatizar os diversos testes. Os diversos testes efetuados tiveram como principais objetivos os seguintes aspetos:

- **Performance:** Efetuando testes às frameworks em máquinas com diferentes recursos computacionais. Foram utilizados 3 diferentes tipos de ambientes de execução, sendo que os principais fatores que se fizeram variar foram a memória RAM, CPU e disco. Este tipo de testes permite verificar o quão mais rápido se torna a execução de uma framework aumentando-se progressivamente a memória RAM disponível, assim como outros recursos computacionais.

- **Escalabilidade:** Verificando os tempos de execução e recursos usados para datasets de diferentes tamanhos. Para este efeito usaram-se duas variantes do dataset MNIST, sendo que a primeira consistiu no dataset original e a segunda em metade desse mesmo dataset.

- **Overhead causado pela privacidade:** Tendo como base um modelo desenvolvido em TensorFlow sem recurso a qualquer framework de privacidade, consegue-se verificar a perda de accuracy, assim como o overhead de tempo de treino causados pela introdução de segurança dos dados.
- **Comparação entre frameworks:** Através das métricas avaliadas, verificar qual ou quais as frameworks que conseguem garantir privacidade com um menor número de recursos, assim como um menor tempo de treino e uma maior accuracy dos modelos produzidos.
- **Variação de parâmetros:** Variar os valores de certos parâmetros como o número de épocas e de batch size pode resultar em conclusões que ilustram o comportamento das diversas frameworks.

Todo o código das frameworks usadas, documentação de instalação e scripts de testes, encontram-se presentes no repositório `https://github.com/barreira/lei-1819`.

3.1 Métricas

Como métricas relativas aos recursos computacionais usados pelas diferentes frameworks, avaliou-se a percentagem de CPU usada, a quantidade de memória RAM, operações IO em disco e, numa fase inicial, avaliou-se para a framework de FL o tráfego na rede. Todas estas métricas foram obtidas com recurso ao comando **dstat** através das flags: -m –cpu –net –disk. Cada uma destas métricas foi avaliada com um intervalo de 60 segundos. As flags mencionadas devolvem vários parâmetros, em particular para o CPU e memória RAM, sendo que destes dois apenas foram usadas as métricas relativas à percentagem de CPU usada por processos de um utilizador (isto é, a execução das frameworks) e da memória RAM apenas se considerou a métrica relativa à quantidade usada.

As restantes métricas dizem respeito à accuracy obtida e ao tempo de execução de uma framework. Os tutoriais usados como exemplo das frameworks já fornecem a métrica relativa à accuracy (com exceção da framework TF Encrypted). Para a medição do tempo usou-se a biblioteca time em Python.

3.2 Ambientes de Execução

Como ambientes de execução, foram disponibilizadas um conjunto de 5 máquinas virtuais do Departamento de Informática (DI) da Universidade do Minho para a paralelização dos testes a efetuar. Todas estas 5 máquinas continham os mesmos recursos computacionais. Para além disso, e por forma a se conseguir verificar como as frameworks se comportam com diferentes recursos, usaram-se mais duas máquinas locais com recurso a virtual machines. Considere-se que estes 3 ambientes de execução distintos se chamam ambiente DI-4 GB (máquinas do DI), 4 GB (uma das máquinas locais com 4 GB de memória RAM) e 8 GB (máquina local com 8 GB de memória RAM).

Para o ambiente de execução DI-4 GB o hardware é o seguinte (resultados obtidos através da execução dos comandos **lshw -short** e **df -H**):

- **CPU:** Dois processadores AMD Opteron 23xx (Gen 3 Class Opteron)
- **RAM:** 4 GB
- **Disco:** 13 GB
- **SO:** CentOS 7 32 bits

Para o ambiente de execução de 4 GB criou-se uma máquina virtual com recurso à plataforma VMWare com 4 GB de memória RAM, 1 processador com 2 cores e com 32 GB de disco SSD. As características base da máquina local são as seguinte:

- **CPU:** 2.3 GHz Intel Core i5
- **RAM:** 8 GB 2133 MHz LPDDR3
- **GPU:** Intel Iris Plus Graphics 640 1535 MB
- **SSD:** 128 GB

Finalmente, para o ambiente de execução 8 GB, também foi criado um ambiente virtual com recurso ao VMWare com 8 GB de memória RAM, 1 processador com 2 cores e com 100 GB de disco SSD. As características do hardware são as seguintes:

- **Central Processing Unit (CPU)**: Intel® Core™ i7-67000HQ 2.60 GHz
- **Random Access Memory (RAM)**: DDR4 2133 MHz SDRAM, 32 GB, (16 GB x 2)
- **Graphics Processing Unit (GPU)**: NVIDIA® GeForce® GTX980M com 8GB GDDR5
- **SSD**: 256 GB

As virtual machines criadas para os ambientes 4 GB e 8 GB, incorporam o sistema operativo CentOS 7, tal como o ambiente DI-4 GB.

3.3 Workloads

Na definição dos workloads a usar para cada framework, houve dois aspetos tidos em consideração: a escolha de datasets apropriados e a definição de modelos de redes neuronais que garantissem uma boa qualidade dos testes.

3.3.1 Escolha dos datasets

A escolha dos datasets baseou-se no estudo teórico efetuado às frameworks de privacidade, optando-se por efetuar testes para datasets conhecidos e de fácil acesso. Ao analisar a tabela 2 verifica-se que o dataset MNIST foi bastante usado como ponto de referência para os testes efetuados pelos desenvolvedores. Por este motivo e aliado ao facto de todas as frameworks escolhidas para o estudo prático fornecerem tutoriais de uso relativos a este dataset, decidiu-se incorporar este dataset nos workloads desenvolvidos.

Muitos dos datasets estudados pelos desenvolvedores são de uma dimensão considerável, o que poderia tornar o tempo de execução dos testes efetuados bastante extensa, tendo por isso sido descartados. Isto acontece para os datasets SVHN e VGG16. Por outro lado, não foram encontrados exemplos de uso das diversas frameworks para outros datasets menos conhecidos como o Glyph, Boston Housing e Pima Indian Diabetes, tendo por esse motivo sido retirados deste estudo.

De forma a testar o comportamento das frameworks para datasets de menores dimensões, optou-se por criar uma variante do MNIST que consistia em metade deste dataset.

3.3.2 Definição dos Modelos

Foi definido apenas um modelo para cada framework para as duas variantes do dataset MNIST. Antes de se efetuar qualquer processo de treino, é realizado um pré-processamento adicional para o caso de se pretender carregar apenas metade do dataset MNIST.

De modo a se obter resultados relativos ao overhead introduzido pela proteção dos dados, criou-se um modelo em TensorFlow com ausência de privacidade dos dados. Este modelo foi criado seguindo o tutorial oficial do TensorFlow para o dataset MNIST (`https://www.tensorflow.org/tutorials`). A definição de todos os modelos criados foi a seguinte:

- **Modelo base do TensorFlow:** Este modelo consiste num modelo Sequential com uma camada de input de 784 neurónios, seguida de uma camada de 512 neurónios, terminando numa camada de output de 10 neurónios. A segunda camada contém a função de ativação ReLU e é aplicado a esta camada um dropout de 20%.

- **TensorFlow Privacy:** Inicialmente o modelo consistiu numa camada convolucional 2D com valores respetivos ao tuplo (16, 8, strides=2, padding='same', activation='relu'), aplicada a uma camada de input de 784 neurónios. A esta camada convolucional foi aplicada uma MaxPool2D(2, 1) seguida de outra camada convolucional de valores (32, 4, stride=2, padding='valid', activation='relu'), sendo também aplicada a esta última camada uma MaxPool2D(2, 1). À saída desta camada é aplicada uma camada com 32 neurónios seguindo uma função de ativação ReLU, sendo finalmente a camada de output uma camada de 10 neurónios.

Este modelo desenvolvido revelou-se bastante complexo, sendo que cada teste efetuado a este modelo poderia demorar mais de 6 horas de execução. Face à grande diversidade de testes projetados, revelou-se inviável efetuar testes para tal modelo. Assim, e tendo como base que esta framework opera sobre modelos construídos em TensorFlow, adaptou-se este modelo para o modelo base do TensorFlow descrito acima. Este passo efetuou-se com a garantia que não era comprometida a privacidade dos dados provenientes do dataset MNIST, já que todo o processo de segurança de dados efetuado por esta framework ocorre após a definição do modelo. Além disso, torna-se possível comparar de uma forma mais justa o overhead causado por esta framework, já que o modelo base do TensorFlow consegue atingir uma accuracy bastante elevada em cerca de um minuto de execução.

- **TF Encrypted:** Esta framework possui um mecanismo próprio de construção de modelos, sendo por esse motivo bastante difícil de avaliar com precisão os diversos parâmetros do modelo proposto. Procedeu-se, no entanto, à análise do excerto de código correspondente à definição dos parâmetros iniciais do modelo:

```
w0 = tf.Variable(tf.random_normal([28 * 28, 512]))
b0 = tf.Variable(tf.zeros([512]))
w1 = tf.Variable(tf.random_normal([512, 10]))
b1 = tf.Variable(tf.zeros([10]))
```

  Tendo em conta o excerto de código ilustrado acima, o modelo descrito assimila-se ao modelo base do TensorFlow, não sendo definidas funções de ativação nem qualquer tipo de dropout.

- **PySyft:** O modelo desenvolvido para esta framework segue um rede neuronal convolucional composta por uma camada convolucional 2D com 1 canal de input, 20 canais de output, kernel size igual a 5 e com um valor de stride igual a 1. A esta camada é aplicada a função de ativação ReLU e uma max pooling 2D. À saída desta camada é aplicada outra camada convolucional composta por 20 canais de input, 50 canais de output, kernel size igual a 5 e stride igual a 1, contendo também a função de ativação ReLU e sendo aplicada max pooling 2D. O resultado é transmitido para mais duas camadas de 800 e de 500 neurónios, respetivamente. A primeira dessas duas camadas contém a função de ativação ReLU e a segunda a função LogSoftmax.

Tentou-se adaptar os modelos acima de modo a que todos estes fossem idênticos para todas as frameworks testadas, aumentando assim a qualidade dos testes efetuados. No entanto, as frameworks TF Encrypted e PySyft possuem implementações próprias para a definição dos modelos, pelo que apenas se conseguiu uniformizar a framework TensorFlow Privacy com o modelo base.

3.4 Scripts e Testes Efetuados

Para uma boa qualidade do estudo comparativo das frameworks de privacidade, foram criados um conjunto de testes para cada um dos ambientes de execução propostos (secção 3.2). Cada framework é acompanhada de dois workloads, sendo que um dos workloads corresponde à totalidade do dataset MNIST e o segundo workload a metade desse mesmo dataset. Além disso, construiu-se um conjunto de testes associados a certos parâmetros dos modelos como o número de épocas e o valor de batch size.

Relativamente ao parâmetro batch size, seguiu-se um estudo ([28]) que verificou que valores baixos de batch size traduziam-se numa accuracy mais baixa quando comparados com valores superiores. Convenientemente, um dos datasets usados como caso de estudo foi o dataset MNIST. Foram testados valores de batch size compreendidos entre 16 e 1024, sendo que os melhores resultados foram obtidos para os valores 512 e 1024 e os piores resultados para os valores 16 e 32. Nos testes efetuados foram utilizados os valores 512 e 1024, assim como os valores 128 e 256, que também forneceram bons resultados nesse mesmo estudo.

Quanto ao número de épocas, não foi encontrada uma concordância sobre os melhores valores para se efetuar um estudo dos modelos produzidos. No entanto, o modelo base do TensorFlow escolhido como referência consegue atingir uma accuracy de 98% em apenas 5 épocas. Assim, tornou-se interessante incluir este valor na pesquisa efetuada neste projeto de modo a se conseguir comparar os resultados obtidos das frameworks face ao modelo base. Como se esperavam tempos de execução bastante elevados (o que acabou por se verificar) optou-se por não escolher valores muito elevados do número de épocas. Posto isto, foi escolhida a gama de valores de 1, 5 e 10 épocas para se efetuar o estudo comparativo.

Tendo em conta os aspetos mencionados, a quantidade de testes efetuada foi de: 3 ambientes de execução * 4 frameworks (incluindo o modelo base) * 2 datasets * 3 valores de épocas * 4 valores de batch size, o que dá um total de 288 testes distintos. Face à grande dimensão da quantidade de testes, houve a necessidade de criação de scripts de automatização que permitissem uma cómoda execução dos diversos testes. De notar que, para uma melhor precisão do estudo de comparação efetuado, cada teste foi executado 3 vezes (o que corresponde a um total de 864 testes individuais). As seguintes secções descrevem, com detalhe, todos os processos efetuados na construção dos scripts de teste, normalização e obtenção dos resultados finais.

3.4.1 Divisão do Dataset MNIST

Os processos efetuados para a divisão do dataset MNIST consistiram em identificar as instruções relativas à obtenção dos convencionais *X_train* e *y_train* e proceder às suas divisões para metade. Enunciam-se, de seguida, os passos efetuados para cada framework.

Modelo base do TensorFlow. Para este modelo (presente no website `https://www.tensorflow.org/tutorials`), apenas foi necessário dividir os dados de treino após a sua normalização, através das seguintes instruções:

```
x_train = x_train[:len(x_train) // 2]
y_train = y_train[:len(y_train) // 2]
```

TensorFlow Privacy. O modo de construção deste modelo é muito semelhante ao modelo base do TensorFlow. Posto isto, efetuaram-se exatamente as mesmas instruções de divisão do dataset para o modelo base, ao resultado proveniente da função *load_mnist()* (repositório `https://github.com/tensorflow/privacy/blob/master/tutorials/mnist_dpsgd_tutorial.py`). Além disso, alteraram-se certos valores, como o número de steps por época, o tamanho da população e a probabilidade da seleção, ajustando esses valores às novas dimensões do dataset.

TF Encrypted. Esta framework divide o processo de construção e de treino do modelo em vários ficheiros. O download e pré-processamento do dataset é efetuado num ficheiro chamado "download.py". Aqui foi efetuada uma divisão dos dados de treino tal como os casos anteriores. O ficheiro "run.py" diz respeito ao treino e execução do modelo, em que apenas foi necessário ajustar o valor do número de iterações face ao novo tamanho do dataset (`https://github.com/tf-encrypted/tf-encrypted/tree/master/examples/mnist`).

PySyft. O modo de funcionamento e de construção de modelos desta framework é completamente distinto das outras três frameworks já que opera em PyTorch. Além disso, esta framework requer a conversão apropriada do dataset para um tipo de dados chamado federated dataset. Devido a estes aspetos, houve uma grande dificuldade de se conseguir dividir o dataset para a construção de um modelo nesta framework, tendo-se abandonado os testes para metade do dataset para o PySyft. Descrevem-se algumas tentativas efetuadas para o processo de divisão do dataset (exemplo do MNIST completo em `https://github.com/OpenMined/PySyft/blob/dev/examples/tutorials`):

- **Uso do syft.BaseDataset em conjunto com TensorFlow:** Nesta tentativa, optou-se por efetuar o download do dataset através do TensorFlow e proceder-se à divisão do mesmo tal como nas frameworks anteriores. Este processo foi realizado, já que os datasets provenientes da TorchVision do PyTorch não são de uma divisão trivial. No entanto, como a versão do PySyft para o dataset completo esperava carregar para o modelo um dataset no formato proveniente da TorchVision, foi necessário alterar esta solução. Assim, seguiu-se o tutorial "Federated Dataset" presente em `https://github.com/OpenMined/PySyft/blob/dev/examples/tutorials/advanced` com recurso à componente BaseDataset da biblioteca syft. Esta componente permite carregar um dataset customizado para o modelo desenvolvido. O dataset foi dividido a meio e

carregado com sucesso para o modelo, no entanto, o modelo desenvolvido esperava uma input shape completamente diferente da shape fornecida pelo dataset, pelo que esta tentativa revelou-se sem sucesso.

- **Uso de SubsetRandomSampler:** Neste processo, tentou-se dividir o dataset original proveniente do TorchVision com recurso a uma seleção aleatória de um conjunto de índices do dataset (neste caso metade dos índices). A conversão para federated dataset foi apenas efetuada após o seguinte processo:

```
batch_size = 16
test_split = .5
shuffle_dataset = True
random_seed= 42

# Creating data indices for training and validation
    splits:
dataset_size = len(trainset)
indices = list(range(dataset_size))
split = int(np.floor(test_split * dataset_size))
if shuffle_dataset :
    np.random.seed(random_seed)
    np.random.shuffle(indices)
train_indices, test_indices = indices[split:],
    indices[:split]

# Creating PT data samplers and loaders:
train_sampler = SubsetRandomSampler(train_indices)
```

Tendo os dados de treino seria suposto fornecer estes dados como sampler a ser retirada do dataset original para o processo de conversão de federated dataset. A função executada foi a seguinte:

```
train_loader =
    sy.FederatedDataLoader(
        trainset.federate((bob, alice)),
        batch_size=args.batch_size,
        sampler=train_sampler)
```

No entanto, foi recebida uma mensagem indicando que a funcionalidade "sampler" não era suportada, tendo todo este processo sido ignorado. Esta tentativa efetuada evidencia certas funcionalidades que ainda não foram implementadas na framework em processo de desenvolvimento.

- **syft.BaseDataset através do dataset MNIST csv:** A última tentativa efetuada consistiu na repetição do uso da componente BaseDataset, mas desta vez aplicada ao dataset MNIST em formato csv (`https:`

//www.kaggle.com/oddrationale/mnist-in-csv). A este dataset apenas foi efetuado o seguinte pré-processamento de modo a se efetuar a divisão e conversão para torch tensors compatíveis com o BaseDataset.

```
df = pd.read_csv('mnist_train.csv')

y = df['label'].values
X = df.drop(['label'], 1).values
X = X[:len(X) // 2]
y = y[:len(y) // 2]

torch_X_train = torch.from_numpy(X).type(torch.
    LongTensor)
torch_y_train = torch.from_numpy(y).type(torch.
    LongTensor)

base = sy.BaseDataset(torch_X_train, torch_y_train)
```

No entanto, o resultado foi análogo à primeira tentativa em que o modelo não esperava a input shape proveniente do dataset processado.

Tendo em conta as três tentativas efetuadas para o processo de divisão do dataset para a framework PySyft, e sabendo que o dataset MNIST original proveniente da TorchVision contém um formato que impede a sua divisão em qualquer editor de texto, abandonou-se a hipótese de se efetuar um estudo desta framework para metade deste dataset.

3.4.2 Scripts de Automatização

Tal como foi dito, a quantidade de testes propostos para o estudo de comparação entre as frameworks é de uma dimensão considerável. Como tal, houve a necessidade de criação de scripts que permitissem, de uma forma cómoda, a execução destes testes. Assim, para cada framework, criou-se um script que permite a sua execução aplicada a um dataset (MNIST full ou MNIST half) para cada valor de batch size e de número de épocas proposto. O resultado consiste num conjunto de diretorias cujos nomes descrevem os testes efetuados. Os nomes dessas diretorias variam entre "mnist-e1-b128" e "mnist-e10-b1024", sendo que a primeira corresponde ao teste efetuado para uma época e batch size 128 e a segunda 10 épocas com batch size 1024.

Os scripts começam por declarar dois arrays com os valores de batch size e de número de épocas a estimar. De seguida, são efetuados dois ciclos que percorrem todos os valores desses dois arrays. A cada iteração é verificado se a diretoria de output dos resultados já existe, sendo esta criada caso contrário. São alterados os valores do número de épocas e de batch size a cada iteração dos dois ciclos através do comando **sed** que permite detetar padrões em ficheiros de texto. Este comando foi envolvido em duas funções chamadas *set_epochs*() e *set_batch_size*(). Como exemplo deste comando, considere-se a

framework TensorFlow Privacy em que o número de épocas é definido pela instrução "EPOCHS = x" e o valor de batch size pela instrução "BATCH_SIZE = y", sendo que x e y correspondem, respetivamente, a valores inteiros de número de épocas e de batch size. Para o exemplo em questão, as funções *set_epochs*() e *set_batch_size*() apresentam as seguintes definições:

```
set_epochs() {
        sed -i -e "s/EPOCHS = [0-9]*/EPOCHS = $1/g" "./
            mnist.py"
}

set_batch_size() {
        sed -i -e "s/BATCH_SIZE = [0-9]*/BATCH_SIZE = $1/
            g" "./mnist.py"
}
```

Após a verificação da diretoria de output dos resultados e de alteração dos ficheiros de código a executar, procede-se à execução do comando **dstat** em background. É também executado o ficheiro de código da framework direcionando-se os seus stardard output e standard error para um ficheiro com a extensão ".txt". Após a execução da framework, é terminado o processo do **dstat** através do comando **pkill**. O resultado do **dstat** é guardado num ficheiro csv. Tanto o ficheiro txt como o ficheiro csv possuem nomes relativos às timestamps de início e de final da execução. O pseudocódigo relativo a todos estes processos descritos é o seguinte:

```
log_dir = ""
for i=1,5,10 do
    set_epochs(i)
    for j=128,256,512,1024 do
        set_batch_size(j)
        log_dir = "mnist-e" + i + "-b" + j

        if !log_dir then
            mkdir -p log_dir

        start = current_date()
        execute_dstat()
        execute_framework()
        pkill dstat
        end = current_date()
        create_files(start, end)
```

Estes scripts descritos apresentam os nomes "run_mnist.sh" para o dataset MNIST full e "run_mnist.sh" para o dataset MNIST half. Foram criadas estas duas versões para cada framework estudada. Além disso, criaram-se dois scripts adicionais chamados "run_mnist_tmux.sh" e "run_mnist_half_tmux.sh" que criam, respetivamente, sessões no terminal com recurso ao programa **tmux**

para os dois scripts mencionados acima. Este tipo de execução foi bastante útil principalmente para os ambientes de execução alojados nas máquinas do DI.

Finalmente, como os ficheiros csv e txt continham nos seus nomes o caratere ":" proveniente das timestamps de início e de final da execução dos scripts, fazia com que estes ficheiros não fossem compatíveis com certos sistemas operativos como o Windows. Posto isto, foi criado um script chamado "rename_logs.py" que renomeia todos os ficheiros de uma determinada diretoria, substituindo o caratere ":" pelo caratere "_".

3.4.3 Normalização dos Resultados

Tendo todos os ficheiros relativos aos resultados de todos os testes efetuados, foi necessário verificar se estes continham informações relativas às métricas que se propôs avaliar. Os ficheiros csv resultantes da execução do comando **dstat** continham todas as métricas associadas aos recursos computacionais usados durante a execução dos testes, pelo que não necessário efetuar qualquer processamento a estes dados. Quanto aos ficheiros txt, esperava-se que estes contivessem informações acerca do tempo de execução e da accuracy obtida. Como já foi mencionado, a informação relativa à métrica do tempo de execução foi obtida através da função *time*() em Python, pelo que todos os ficheiros txt continham esta informação. Finalmente, todas as frameworks informavam a accuracy obtida, com a exceção da framework TF Encrypted. Esta framework efetuava várias previsões do modelo e dava como output dois arrays para cada previsão. Um array com as classes esperadas e outro array com as classes obtidas. O formato do resultado da execução desta framework era o seguinte:

```
Expect [2 2 6 3 2 6 5 4 8 9 7 1 3 0 3 8 3 1 9 3 4 4 6 4 2 1 8
2 5 4 8 8 4 0 0 2 3 2 7 7 0 8 7 4 4 7 9 6 9 0 9 8 0 4 6 0 6 3
5 4 8 3 3 9 3 3 3 7 8 0 8 2 1 7 0 6 5 4 3 8 0 9 6 3 8 0 9 9 6
8 6 8 5 7 8 6 0 2 4 0 2 2 3 1 9 7 5 1 0 8 4 6 2 6 7 9 3 2 9 8
2 2 9 2 7 3 5 9 1 8 0 2 0 5 2 1 3 7 6 7 1 2 5 8 0 3 7 2 4 0 9
1 8 6 7 7 4 3 4 9 1 9 5 1 7 3 9 7 6 9 1 3 7 8 3 3 6 7 2 8 5 8
5 1 1 4 4 3 1 0 7 7 0 7 9 4 4 8 5 5 4 0 8 2 1 0 8 4 5 0 4 0 6
1 7 3 2 6 7 2 6 9 3 1 4 6 2 5 4 2 0 6 2 1 7 3 4 1 0 5 4 3 1 1
7 4 9 9 4 8 4 0 2 4 5 1]
Result [2 2 2 4 2 6 5 4 2 4 7 4 5 0 5 5 2 1 4 2 4 4 2 4 2 5 4
2 5 4 4 4 4 5 0 2 5 2 4 4 5 5 7 4 4 7 9 6 4 0 4 5 0 4 2 0 5 4
5 4 4 5 5 9 3 3 7 7 8 0 2 2 4 7 0 6 5 4 5 5 0 7 5 5 8 0 9 9 6
2 6 5 5 2 5 6 0 2 4 0 2 2 5 2 4 7 5 5 0 5 4 6 2 6 7 4 9 4 4 8
2 2 9 2 7 5 5 9 1 5 5 2 0 5 2 1 3 7 5 7 1 2 5 8 0 5 4 9 4 0 9
1 5 6 7 5 4 5 4 9 1 4 5 1 7 3 4 7 6 9 4 5 2 5 3 5 5 4 2 4 5 5
5 1 1 4 4 5 1 0 7 7 0 7 4 4 4 2 5 5 4 0 5 2 5 4 8 4 5 0 4 0 4
1 4 2 2 5 5 2 6 4 5 1 4 6 2 5 4 2 0 5 2 1 7 3 4 5 0 5 4 3 1 4
7 4 4 4 4 5 4 0 2 4 5 1]
```

Figura 1: Resultados obtidos pela framework TF Encrypted

Face a este resultado completamente distinto das outras frameworks, foi construído um script em Python chamado "accuracy.py" que percorre todos os

ficheiros txt presentes numa diretoria de resultados de execução. Para cada ficheiro são extraídos os conjuntos dos dois arrays de resultados esperados e resultados obtidos. Estes dois arrays são comparados e é calculado o rácio de classes corretamente classificadas pelo array de resultados obtidos. O resultado final obtido consiste na accuracy do modelo, sendo este resultado acrescentado no final do ficheiro txt que lhe deu origem.

3.4.4 Resultados Finais

A informação resultante traduzida de todos os testes efetuados e suas múltiplas execuções resultou num conjunto de mais de 1700 ficheiros, o que se torna bastante inviável para se efetuar uma análise manual. Posto isto, foram criados scripts para cada framework que agrupam estes resultados num formato compreensível. Todos estes scripts executam a mesma tarefa para cada framework, sendo apenas distintos na leitura dos resultados obtidos pelas mesmas, já que os ficheiros txt obtidos apresentam diferentes formatos.

Como foi falado nas secções anteriores, os resultados são divididos em diversas diretorias. Uma framework apresenta, para um dataset, diretorias que contêm os resultados dos testes efetuados às diversas combinações entre o número de épocas e batch size estimados. Assim, para cada dataset é criada uma tabela que contém informações de todos estes testes. Cada linha dessa tabela corresponde a um teste para uma combinação desses valores (e.g. 5 épocas e batch size igual a 256), apresentando a média e o desvio padrão dos resultados obtidos das diversas execuções desse teste. Cada ficheiro csv é processado segundo a estrutura de dados **DataFrame** da biblioteca pandas, onde são apenas retidas informações acerca das métricas de memória RAM usada, carga de CPU derivada dos processos do utilizador, e operações de leitura e escrita em disco. Dos ficheiros txt, são extraídas as métricas relativas ao tempo de execução e da accuracy, dependendo dos seus formatos. Estas duas métricas são acrescentadas à **DataFrame**, sendo esta estrutura de dados convertida para um ficheiro csv que contém os resultados finais. O pseudocódigo do algoritmo desenvolvido é o seguinte:

```
results = pd.DataFrame(columns=['test', 'used', 'usr', '
    read', 'writ', 'acc', 'time'])

for subdir in test_dir do
    data_subdir = pd.DataFrame(columns=['used', 'usr', '
        read', writ]
    time = []
    acc = []

    for file in subdir do
        if file is csv do
            add_csv_to_dataframe(data_subdir)
        else do
            # file is txt
```

```
            addAccurayToArray(file, acc)
            addTimeToArray(file, time)

        # Calculate standard deviation and average

        acc_avg = avg(acc)
        time_avg = avg(time)
        acc_std = std(acc)
        time_avg = std(time)
        std_data_subdir = std(data_subdir)
        avg_data_subdir = avg(data_subdir)

        # Test name is in subdirectory name
        test_name = get_test_name(subdir)

        # Add all this data to results DataFrame
        ...

convert_to_csv(results)
```

Com foi mencionado na secção 3.1, considerou-se interessante numa fase inicial, avaliar o tráfego na rede para a framework PySyft, já que esta assenta no algoritmo de federated learning (2.1.2). No entanto, após uma leitura mais aprofundada da documentação dos tutoriais desta framework, verificou-se que esta componente mais realista da framework não estava a ser usada. Ao invés, foi efetuada uma simulação de diversos dispositivos com recurso à componente **VirtualWorkers**. Devido a este aspeto, foram descartadas as métricas relativas ao tráfego na rede.

4 Resultados Obtidos

Nesta secção, apresentam-se todos os resultados obtidos dos processos efetuados e descritos na secção anterior. São apresentadas diversas tabelas que representam os valores de todos os testes efetuados. Cada linha de cada tabela contém as médias e os desvios padrão das diferentes métricas avaliadas para cada teste. A legenda das diversas tabelas é a seguinte:

- **test:** Nome do teste efetuado, indicando o número de épocas e valor de batch size testado. Por exemplo, "e1-b1024" corresponde a um teste efetuado para uma época e batch size igual a 1024.
- **used:** Quantidade de memória RAM usada em megabytes (base 10).
- **usr:** Percentagem de CPU usado por processos do utilizador.
- **read:** Leitura em disco em kilobytes (base 10).
- **writ:** Escrita em disco em kilobytes (base 10).

- **acc:** Accuracy do modelo (entre 0 e 1).
- **time:** Tempo de execução em segundos.

Os resultados obtidos são ilustrados pelas seguintes tabelas. De notar que, tal como foi mencionado na secção 3.4.1, não são apresentados resultados para a framework PySyft quando aplicada ao dataset MNIST half.

Para uma melhor interpretação dos resultados obtidos, esta é dividida em duas partes. Na primeira parte enumeram-se um conjunto de observações mais gerais derivadas dos resultados obtidos. Na segunda parte é efetuada uma análise mais detalhada dos resultados obtidos para cada framework acompanhada de uma possível justificação com base nas suas características e algoritmos.

Tabela 3: TensorFlow Privacy no ambiente de execução de DI-4 GB e com recurso ao dataset MNIST full

test	used	usr	read	writ	acc	time
e1-b128	896.27 +/- 221.34	70.98 +/- 25.47	41.02 +/- 216.65	175.28 +/- 669.46	0.86 +/- 0.00	540.34 +/- 12.43
e1-b256	892.56 +/- 229.25	74.52 +/- 27.94	0.08 +/- 0.29	55.09 +/- 148.66	0.84 +/- 0.01	459.94 +/- 17.17
e1-b512	894.06 +/- 239.84	75.98 +/- 30.61	0.09 +/- 0.30	120.79 +/- 307.81	0.75 +/- 0.03	409.55 +/- 8.83
e1-b1024	888.04 +/- 241.38	77.27 +/- 32.01	3.80 +/- 16.96	106.36 +/- 274.10	0.60 +/- 0.02	388.95 +/- 9.95
e5-b128	973.51 +/- 165.79	73.26 +/- 15.36	2.49 +/- 30.92	89.63 +/- 189.49	0.90 +/- 0.00	3105.42 +/- 67.78
e5-b256	972.45 +/- 166.16	79.40 +/- 16.64	1.26 +/- 13.29	99.58 +/- 183.34	0.90 +/- 0.00	2445.73 +/- 73.20
e5-b512	981.25 +/- 168.53	82.64 +/- 16.47	0.99 +/- 10.14	114.88 +/- 226.71	0.85 +/- 0.03	2197.95 +/- 38.40
e5-b1024	993.25 +/- 168.55	85.07 +/- 16.64	0.05 +/- 0.24	107.44 +/- 206.28	0.83 +/- 0.01	2044.86 +/- 28.45
e10-b128	1005.91 +/- 146.50	70.35 +/- 15.38	1.13 +/- 10.56	72.53 +/- 161.03	0.91 +/- 0.00	7296.73 +/- 68.94
e10-b256	1001.98 +/- 150.40	76.82 +/- 16.41	1.23 +/- 12.61	90.97 +/- 180.55	0.91 +/- 0.00	5492.93 +/- 96.77
e10-b512	1011.12 +/- 152.53	80.59 +/- 16.06	3.24 +/- 44.71	102.62 +/- 196.06	0.91 +/- 0.00	4828.25 +/- 38.26
e10-b1024	1082.57 +/- 159.23	83.34 +/- 15.41	10.44 +/- 103.12	113.11 +/- 213.08	0.88 +/- 0.03	4422.14 +/- 67.89

Tabela 4: TensorFlow Privacy no ambiente de execução de DI-4 GB e com recurso ao dataset MNIST half

test	used	usr	read	writ	acc	time
e1-b128	815.04 +/- 246.74	64.17 +/- 31.45	4.81 +/- 18.89	93.16 +/- 197.64	0.81 +/- 0.01	286.91 +/- 10.51
e1-b256	790.18 +/- 259.08	65.52 +/- 35.00	0.08 +/- 0.16	189.86 +/- 373.66	0.76 +/- 0.02	237.49 +/- 9.96
e1-b512	781.65 +/- 271.11	65.07 +/- 38.66	0.09 +/- 0.17	149.54 +/- 286.62	0.59 +/- 0.04	216.82 +/- 6.92
e1-b1024	793.12 +/- 277.52	66.22 +/- 39.41	0.09 +/- 0.17	185.30 +/- 348.02	0.45 +/- 0.04	206.37 +/- 6.98
e5-b128	921.83 +/- 165.62	71.40 +/- 17.54	5.96 +/- 53.74	159.31 +/- 250.50	0.89 +/- 0.00	1615.50 +/- 23.73
e5-b256	910.35 +/- 171.31	76.33 +/- 19.34	0.02 +/- 0.10	196.02 +/- 243.14	0.86 +/- 0.03	1312.76 +/- 24.48
e5-b512	928.52 +/- 175.68	78.56 +/- 20.46	2.17 +/- 15.09	180.24 +/- 216.48	0.88 +/- 0.00	1171.99 +/- 25.91
e5-b1024	934.63 +/- 179.37	80.94 +/- 20.38	0.02 +/- 0.09	177.50 +/- 271.44	0.78 +/- 0.03	1107.32 +/- 6.89
e10-b128	966.90 +/- 141.38	69.06 +/- 15.70	0.23 +/- 2.97	131.94 +/- 202.85	0.90 +/- 0.00	3827.58 +/- 70.12
e10-b256	967.19 +/- 144.51	73.99 +/- 17.54	0.02 +/- 0.13	171.20 +/- 214.91	0.90 +/- 0.00	2977.12 +/- 33.26
e10-b512	974.32 +/- 144.13	77.10 +/- 17.86	0.11 +/- 1.21	184.52 +/- 215.73	0.85 +/- 0.02	2624.30 +/- 26.38
e10-b1024	997.40 +/- 147.24	79.22 +/- 18.08	0.48 +/- 5.15	192.84 +/- 230.82	0.79 +/- 0.03	2466.91 +/- 39.48

Tabela 5: TensorFlow Privacy no ambiente de execução de 4 GB e com recurso ao dataset MNIST full

test	used	usr	read	writ	acc	time
e1-b128	905.76 +/- 262.68	57.44 +/- 20.14	18.17 +/- 34.98	298.66 +/- 292.73	0.86 +/- 0.00	145.08 +/- 6.38
e1-b256	816.39 +/- 300.72	54.02 +/- 25.00	26.97 +/- 40.23	517.48 +/- 296.74	0.83 +/- 0.02	107.93 +/- 3.84
e1-b512	817.86 +/- 301.34	55.43 +/- 26.32	26.76 +/- 39.83	659.83 +/- 335.93	0.74 +/- 0.02	84.62 +/- 12.03
e1-b1024	821.26 +/- 303.15	60.07 +/- 31.10	26.60 +/- 39.55	517.08 +/- 294.42	0.61 +/- 0.06	68.39 +/- 3.14
e5-b128	1091.30 +/- 156.69	66.22 +/- 11.84	5.59 +/- 19.69	326.50 +/- 282.43	0.90 +/- 0.00	823.77 +/- 22.76
e5-b256	1216.03 +/- 334.23	70.65 +/- 14.13	5.38 +/- 19.02	346.07 +/- 236.44	0.90 +/- 0.00	547.19 +/- 16.23
e5-b512	1228.07 +/- 322.41	73.10 +/- 15.66	6.27 +/- 19.61	402.56 +/- 239.74	0.87 +/- 0.03	459.77 +/- 7.63
e5-b1024	1242.04 +/- 336.95	77.76 +/- 17.72	7.74 +/- 20.13	536.36 +/- 313.81	0.87 +/- 0.03	403.54 +/- 10.47
e10-b128	1276.58 +/- 224.64	64.00 +/- 11.18	10.26 +/- 94.08	252.75 +/- 263.56	0.91 +/- 0.00	2144.22 +/- 129.24
e10-b256	1359.04 +/- 228.35	69.30 +/- 11.35	8.16 +/- 46.80	327.35 +/- 243.27	0.91 +/- 0.00	1393.91 +/- 54.44
e10-b512	1344.43 +/- 229.95	72.65 +/- 11.65	3.16 +/- 12.62	381.47 +/- 224.83	0.88 +/- 0.04	1091.44 +/- 19.85
e10-b1024	1344.23 +/- 238.60	78.43 +/- 12.36	2.48 +/- 10.58	483.53 +/- 270.47	0.89 +/- 0.01	861.51 +/- 62.95

Tabela 6: TensorFlow Privacy no ambiente de execução de 4 GB e com recurso ao dataset MNIST half

test	used	usr	read	writ	acc	time
e1-b128	730.35 +/- 331.76	40.54 +/- 34.19	231.79 +/- 440.92	500.99 +/- 405.84	0.80 +/- 0.00	72.68 +/- 1.39
e1-b256	652.72 +/- 393.62	25.92 +/- 31.95	67.85 +/- 67.48	451.50 +/- 509.71	0.74 +/- 0.03	55.78 +/- 3.70
e1-b512	469.47 +/- 165.64	11.06 +/- 11.33	89.78 +/- 60.60	451.60 +/- 617.75	0.59 +/- 0.05	45.70 +/- 3.54
e1-b1024	471.25 +/- 165.83	11.41 +/- 11.17	89.21 +/- 60.00	453.90 +/- 614.54	0.42 +/- 0.02	42.09 +/- 2.35
e5-b128	1107.63 +/- 383.83	60.01 +/- 19.60	14.61 +/- 37.41	477.92 +/- 311.85	0.89 +/- 0.00	473.12 +/- 46.90
e5-b256	988.43 +/- 244.69	61.84 +/- 22.18	95.31 +/- 342.51	500.50 +/- 284.87	0.84 +/- 0.03	328.63 +/- 15.44
e5-b512	956.37 +/- 280.17	62.86 +/- 24.48	17.09 +/- 41.85	560.19 +/- 287.36	0.85 +/- 0.03	266.90 +/- 12.80
e5-b1024	1094.15 +/- 471.83	64.78 +/- 27.34	40.93 +/- 68.21	679.23 +/- 359.16	0.76 +/- 0.07	228.91 +/- 15.09
e10-b128	1209.73 +/- 366.76	61.53 +/- 12.13	9.45 +/- 43.10	446.47 +/- 233.47	0.90 +/- 0.00	1087.01 +/- 74.10
e10-b256	1223.22 +/- 379.45	65.34 +/- 13.38	16.72 +/- 41.04	518.15 +/- 199.35	0.90 +/- 0.00	751.91 +/- 28.93
e10-b512	1336.83 +/- 345.48	66.31 +/- 13.68	58.56 +/- 299.44	612.26 +/- 240.44	0.87 +/- 0.03	624.33 +/- 2.01
e10-b1024	1235.97 +/- 313.30	70.13 +/- 15.72	7.11 +/- 23.76	651.93 +/- 272.42	0.87 +/- 0.03	552.22 +/- 27.22

Tabela 7: TensorFlow Privacy no ambiente de execução de 8 GB e com recurso ao dataset MNIST full

test	used	usr	read	writ	acc	time
e1-b128	1396.39 +/- 287.95	81.12 +/- 27.56	1663.37 +/- 3267.82	232.07 +/- 209.85	0.85 +/- 0.00	151.31 +/- 2.50
e1-b256	1410.99 +/- 281.67	87.63 +/- 19.66	494.90 +/- 1422.49	328.57 +/- 351.59	0.84 +/- 0.01	123.76 +/- 1.69
e1-b512	1325.94 +/- 308.33	83.48 +/- 21.51	502.75 +/- 1152.17	505.68 +/- 305.15	0.72 +/- 0.02	109.39 +/- 1.55
e1-b1024	1337.80 +/- 317.60	84.18 +/- 21.21	393.99 +/- 886.58	585.52 +/- 327.74	0.58 +/- 0.03	99.99 +/- 0.99
e5-b128	1590.26 +/- 170.70	97.54 +/- 8.56	42.58 +/- 265.53	290.94 +/- 272.28	0.90 +/- 0.00	921.76 +/- 19.06
e5-b256	1596.49 +/- 181.02	97.32 +/- 8.95	23.22 +/- 116.98	332.60 +/- 253.68	0.90 +/- 0.00	708.97 +/- 18.91
e5-b512	1604.00 +/- 195.97	97.12 +/- 9.19	19.71 +/- 86.92	291.08 +/- 202.08	0.86 +/- 0.03	618.25 +/- 19.18
e5-b1024	1645.04 +/- 208.33	96.69 +/- 8.94	67.82 +/- 117.27	301.82 +/- 260.67	0.83 +/- 0.04	575.82 +/- 40.73
e10-b128	1761.44 +/- 119.01	98.99 +/- 4.65	14.59 +/- 68.75	241.01 +/- 238.05	0.91 +/- 0.00	2237.47 +/- 78.44
e10-b256	1758.47 +/- 128.58	98.93 +/- 4.85	4.14 +/- 26.12	304.30 +/- 234.32	0.91 +/- 0.00	1629.86 +/- 39.53
e10-b512	1763.67 +/- 138.43	98.86 +/- 4.89	4.06 +/- 22.66	320.12 +/- 204.95	0.88 +/- 0.04	1362.83 +/- 33.67
e10-b1024	1800.02 +/- 150.74	98.71 +/- 4.86	3.97 +/- 20.44	321.69 +/- 219.77	0.88 +/- 0.03	1233.04 +/- 51.43

Tabela 8: TensorFlow Privacy no ambiente de execução de 8 GB e com recurso ao dataset MNIST half

test	used	usr	read	writ	acc	time
e1-b128	1299.68 +/- 292.43	63.46 +/- 39.74	1301.83 +/- 2849.91	329.00 +/- 207.48	0.80 +/- 0.01	78.10 +/- 1.31
e1-b256	1314.65 +/- 291.82	66.05 +/- 40.09	143.98 +/- 212.78	527.45 +/- 400.51	0.74 +/- 0.03	69.40 +/- 8.04
e1-b512	1165.64 +/- 264.10	60.37 +/- 40.36	168.99 +/- 218.05	462.44 +/- 397.27	0.58 +/- 0.01	58.10 +/- 2.87
e1-b1024	1057.73 +/- 55.89	36.83 +/- 30.98	276.44 +/- 211.10	191.71 +/- 139.18	0.47 +/- 0.03	54.10 +/- 1.75
e5-b128	1552.23 +/- 192.49	91.61 +/- 22.13	32.64 +/- 107.82	395.41 +/- 278.46	0.89 +/- 0.00	474.38 +/- 11.39
e5-b256	1541.41 +/- 206.96	91.13 +/- 21.39	34.18 +/- 100.32	350.30 +/- 167.48	0.88 +/- 0.01	378.42 +/- 3.87
e5-b512	1540.17 +/- 224.69	90.36 +/- 21.14	37.01 +/- 96.53	434.31 +/- 141.53	0.81 +/- 0.05	326.19 +/- 6.48
e5-b1024	1568.47 +/- 240.53	90.36 +/- 20.41	36.06 +/- 91.45	485.61 +/- 205.73	0.78 +/- 0.04	301.68 +/- 6.41
e10-b128	1650.64 +/- 145.74	97.17 +/- 10.88	9.63 +/- 47.03	408.10 +/- 173.42	0.90 +/- 0.00	1166.28 +/- 28.69
e10-b256	1641.48 +/- 159.44	96.91 +/- 10.47	10.39 +/- 43.09	440.19 +/- 175.68	0.88 +/- 0.03	892.28 +/- 12.06
e10-b512	1655.00 +/- 173.53	96.74 +/- 10.19	11.09 +/- 40.62	431.22 +/- 176.88	0.87 +/- 0.04	752.80 +/- 11.11
e10-b1024	1680.77 +/- 185.50	96.59 +/- 9.74	10.75 +/- 38.25	443.82 +/- 168.27	0.88 +/- 0.01	691.71 +/- 4.24

Tabela 9: PySyft no ambiente de execução de DI-4 GB e com recurso ao dataset MNIST full

test	used	usr	read	writ	acc	time
e1-b128	820.92 +/- 132.05	72.31 +/- 26.89	47.00 +/- 213.66	76.67 +/- 381.27	0.92 +/- 0.00	494.41 +/- 2.37
e1-b256	826.26 +/- 147.46	74.76 +/- 29.94	5.35 +/- 24.36	2.45 +/- 4.68	0.89 +/- 0.00	398.73 +/- 7.17
e1-b512	844.10 +/- 142.41	79.81 +/- 28.87	1.90 +/- 8.57	2.56 +/- 5.04	0.83 +/- 0.00	483.77 +/- 8.47
e1-b1024	906.30 +/- 179.32	78.41 +/- 30.44	0.25 +/- 0.68	2.35 +/- 4.59	0.72 +/- 0.00	408.57 +/- 3.33
e5-b128	878.95 +/- 81.96	76.38 +/- 14.41	1.56 +/- 13.93	0.77 +/- 2.48	0.98 +/- 0.00	1828.31 +/- 5.85
e5-b256	869.77 +/- 97.26	76.47 +/- 17.98	0.76 +/- 3.17	2.33 +/- 8.34	0.96 +/- 0.00	1214.32 +/- 14.51
e5-b512	913.40 +/- 83.57	87.94 +/- 15.28	0.06 +/- 0.34	0.72 +/- 2.38	0.94 +/- 0.00	2008.27 +/- 11.03
e5-b1024	911.03 +/- 104.91	84.55 +/- 18.41	0.10 +/- 0.42	1.01 +/- 2.90	0.91 +/- 0.00	1300.59 +/- 11.02
e10-b128	928.84 +/- 80.47	77.29 +/- 10.67	0.03 +/- 0.26	0.55 +/- 1.85	0.98 +/- 0.00	3532.55 +/- 40.12
e10-b256	918.55 +/- 91.75	76.75 +/- 13.82	0.07 +/- 0.33	0.68 +/- 2.29	0.98 +/- 0.00	2193.84 +/- 5.87
e10-b512	958.96 +/- 79.48	89.33 +/- 11.04	0.04 +/- 0.28	0.55 +/- 1.80	0.97 +/- 0.00	3942.88 +/- 45.74
e10-b1024	949.28 +/- 95.74	85.88 +/- 13.55	0.05 +/- 0.31	0.78 +/- 2.31	0.94 +/- 0.00	2462.32 +/- 12.52

Tabela 10: PySyft no ambiente de execução de 4 GB e com recurso ao dataset MNIST full

test	used	usr	read	writ	acc	time
e1-b128	792.39 +/- 326.11	52.37 +/- 22.97	749.85 +/- 1056.37	1891.15 +/- 2844.26	0.92 +/- 0.00	58.09 +/- 6.55
e1-b256	663.63 +/- 204.24	49.73 +/- 23.83	486.10 +/- 689.63	2127.93 +/- 3034.10	0.89 +/- 0.00	50.70 +/- 5.94
e1-b512	515.04 +/- 7.37	45.54 +/- 23.10	605.51 +/- 678.03	2649.31 +/- 2989.32	0.83 +/- 0.00	42.59 +/- 2.17
e1-b1024	515.26 +/- 7.01	46.82 +/- 21.09	571.63 +/- 624.22	2500.65 +/- 2752.82	0.72 +/- 0.00	34.83 +/- 0.95
e5-b128	901.05 +/- 201.31	65.77 +/- 11.98	114.06 +/- 312.96	477.32 +/- 1380.59	0.98 +/- 0.00	254.26 +/- 0.76
e5-b256	897.97 +/- 232.07	65.16 +/- 9.76	108.75 +/- 264.18	474.42 +/- 1154.29	0.96 +/- 0.00	194.27 +/- 2.77
e5-b512	929.12 +/- 250.23	67.75 +/- 9.48	104.20 +/- 221.57	415.48 +/- 976.52	0.94 +/- 0.00	201.48 +/- 9.66
e5-b1024	891.76 +/- 282.80	63.35 +/- 7.42	113.84 +/- 221.05	494.65 +/- 959.27	0.91 +/- 0.00	149.19 +/- 9.73
e10-b128	988.41 +/- 183.50	68.84 +/- 4.97	39.42 +/- 123.69	159.41 +/- 536.24	0.98 +/- 0.00	491.48 +/- 41.96
e10-b256	976.25 +/- 211.69	68.13 +/- 4.43	40.92 +/- 113.28	177.99 +/- 491.06	0.98 +/- 0.00	349.40 +/- 17.27
e10-b512	1027.73 +/- 212.36	71.28 +/- 4.90	31.30 +/- 91.40	136.09 +/- 396.14	0.97 +/- 0.00	390.90 +/- 29.25
e10-b1024	981.16 +/- 243.67	67.94 +/- 3.80	41.64 +/- 94.87	178.60 +/- 411.96	0.94 +/- 0.00	269.12 +/- 6.53

Tabela 11: PySyft no ambiente de execução de 8 GB e com recurso ao dataset MNIST full

test	used	usr	read	writ	acc	time
e1-b128	1928.18 +/- 225.66	77.01 +/- 11.01	56.59 +/- 38.03	115.79 +/- 49.90	0.92 +/- 0.00	47.55 +/- 0.60
e1-b256	1824.56 +/- 11.33	71.68 +/- 2.70	75.33 +/- 7.62	139.76 +/- 13.80	0.89 +/- 0.00	38.07 +/- 0.23
e1-b512	1824.82 +/- 11.71	71.74 +/- 2.68	75.17 +/- 7.58	139.47 +/- 13.74	0.83 +/- 0.00	41.65 +/- 0.93
e1-b1024	1832.99 +/- 3.51	71.80 +/- 2.67	74.99 +/- 7.54	139.16 +/- 13.67	0.72 +/- 0.00	36.72 +/- 0.83
e5-b128	2206.00 +/- 230.55	92.25 +/- 12.35	18.71 +/- 34.00	38.15 +/- 61.12	0.98 +/- 0.00	207.63 +/- 6.29
e5-b256	2160.75 +/- 251.38	89.83 +/- 13.36	24.71 +/- 37.24	52.75 +/- 65.44	0.96 +/- 0.00	148.23 +/- 0.04
e5-b512	2281.31 +/- 274.15	92.13 +/- 12.02	18.40 +/- 33.43	35.86 +/- 61.08	0.94 +/- 0.00	181.44 +/- 0.29
e5-b1024	2205.81 +/- 283.04	89.66 +/- 12.95	24.33 +/- 36.67	47.36 +/- 66.49	0.91 +/- 0.00	141.62 +/- 4.75
e10-b128	2323.62 +/- 225.63	95.27 +/- 9.49	10.36 +/- 26.10	20.87 +/- 47.84	0.98 +/- 0.00	399.99 +/- 2.09
e10-b256	2314.44 +/- 247.00	94.25 +/- 10.12	12.58 +/- 28.11	25.43 +/- 51.25	0.98 +/- 0.00	289.72 +/- 3.29
e10-b512	2424.33 +/- 266.52	95.07 +/- 9.08	10.05 +/- 25.32	20.45 +/- 46.35	0.97 +/- 0.00	361.99 +/- 4.47
e10-b1024	2333.17 +/- 269.56	93.79 +/- 10.38	13.86 +/- 28.80	27.46 +/- 52.68	0.94 +/- 0.00	270.68 +/- 4.00

Tabela 12: TF Encrypted no ambiente de execução de DI-4 GB e com recurso ao dataset MNIST full

test	used	usr	read	writ	acc	time
e1-b128	611.50 +/- 21.05	1.01 +/- 0.05	1.61 +/- 0.02	10.06 +/- 0.10	0.41 +/- 0.04	40.41 +/- 0.55
e1-b256	609.74 +/- 18.77	1.01 +/- 0.05	1.61 +/- 0.02	10.06 +/- 0.10	0.55 +/- 0.03	42.98 +/- 1.25
e1-b512	610.25 +/- 19.56	1.01 +/- 0.05	1.61 +/- 0.02	10.06 +/- 0.10	0.65 +/- 0.08	50.67 +/- 0.47
e1-b1024	812.11 +/- 221.23	33.92 +/- 36.05	0.80 +/- 0.88	6.33 +/- 4.09	0.60 +/- 0.03	70.86 +/- 1.08
e5-b128	733.22 +/- 98.90	35.84 +/- 26.13	0.66 +/- 0.80	6.18 +/- 4.56	0.72 +/- 0.02	168.24 +/- 6.35
e5-b256	739.36 +/- 99.67	36.10 +/- 26.33	0.54 +/- 0.81	5.11 +/- 4.06	0.49 +/- 0.06	159.83 +/- 3.55
e5-b512	739.14 +/- 100.23	37.08 +/- 27.07	0.54 +/- 0.81	4.47 +/- 4.30	0.56 +/- 0.09	166.87 +/- 3.69
e5-b1024	810.25 +/- 150.26	46.07 +/- 27.99	0.40 +/- 0.73	3.78 +/- 4.00	0.47 +/- 0.07	182.94 +/- 1.15
e10-b128	770.37 +/- 76.06	44.81 +/- 20.16	0.27 +/- 0.62	2.22 +/- 3.70	0.66 +/- 0.03	322.71 +/- 13.31
e10-b256	777.26 +/- 79.71	45.15 +/- 20.33	0.27 +/- 0.62	2.40 +/- 3.62	0.56 +/- 0.14	311.66 +/- 11.83
e10-b512	780.61 +/- 82.79	46.60 +/- 21.04	0.27 +/- 0.62	2.30 +/- 3.72	0.53 +/- 0.07	317.05 +/- 9.72
e10-b1024	809.18 +/- 117.93	47.99 +/- 21.87	0.28 +/- 0.61	2.76 +/- 3.79	0.52 +/- 0.09	322.28 +/- 10.38

Tabela 13: TF Encrypted no ambiente de execução de DI-4 GB e com recurso ao dataset MNIST half

test	used	usr	read	writ	acc	time
e1-b128	561.36 +/- 48.05	1.48 +/- 0.98	1.78 +/- 0.29	11.40 +/- 2.13	0.62 +/- 0.05	24.61 +/- 0.54
e1-b256	548.40 +/- 37.74	1.48 +/- 0.98	1.79 +/- 0.30	11.40 +/- 2.13	0.69 +/- 0.08	29.40 +/- 0.81
e1-b512	547.05 +/- 36.03	1.49 +/- 0.98	1.79 +/- 0.29	11.40 +/- 2.13	0.53 +/- 0.09	37.66 +/- 0.88
e1-b1024	652.57 +/- 216.00	19.05 +/- 35.13	3.20 +/- 2.84	11.84 +/- 1.95	0.63 +/- 0.03	56.07 +/- 1.82
e5-b128	641.10 +/- 110.64	27.36 +/- 28.34	17.03 +/- 26.38	7.07 +/- 4.94	0.63 +/- 0.07	87.59 +/- 2.41
e5-b256	642.62 +/- 112.38	26.95 +/- 27.89	0.89 +/- 1.00	6.84 +/- 5.18	0.60 +/- 0.07	87.39 +/- 4.66
e5-b512	644.54 +/- 113.42	27.72 +/- 28.75	0.89 +/- 1.00	6.96 +/- 5.05	0.58 +/- 0.04	91.22 +/- 4.75
e5-b1024	646.37 +/- 115.07	28.74 +/- 29.85	2.58 +/- 3.79	6.98 +/- 5.02	0.68 +/- 0.05	107.57 +/- 3.14
e10-b128	684.72 +/- 100.41	38.35 +/- 25.45	0.54 +/- 0.88	4.67 +/- 4.88	0.65 +/- 0.05	165.97 +/- 9.39
e10-b256	677.12 +/- 102.84	35.84 +/- 25.77	0.60 +/- 0.91	4.83 +/- 5.10	0.59 +/- 0.01	164.31 +/- 9.40
e10-b512	679.56 +/- 104.37	37.01 +/- 26.65	1.00 +/- 1.32	5.02 +/- 4.93	0.64 +/- 0.04	164.35 +/- 2.43
e10-b1024	736.43 +/- 150.33	43.61 +/- 27.71	0.49 +/- 0.85	4.79 +/- 4.65	0.61 +/- 0.04	181.04 +/- 6.60

Tabela 14: TF Encrypted no ambiente de execução de 4 GB e com recurso ao dataset MNIST full

test	used	usr	read	writ	acc	time
e1-b128	582.67 +/- 28.17	16.11 +/- 1.74	194.60 +/- 190.04	144.04 +/- 138.03	0.60 +/- 0.07	17.45 +/- 0.83
e1-b256	582.88 +/- 27.35	16.26 +/- 1.61	193.18 +/- 187.85	143.00 +/- 136.42	0.58 +/- 0.02	17.61 +/- 0.87
e1-b512	582.87 +/- 27.22	16.40 +/- 1.48	191.87 +/- 185.84	142.05 +/- 134.96	0.56 +/- 0.14	20.31 +/- 1.21
e1-b1024	582.20 +/- 26.80	16.59 +/- 1.33	190.38 +/- 183.57	140.98 +/- 133.32	0.67 +/- 0.06	23.75 +/- 1.29
e5-b128	697.67 +/- 128.81	31.30 +/- 15.88	94.36 +/- 154.27	72.02 +/- 111.49	0.60 +/- 0.05	70.85 +/- 3.66
e5-b256	698.06 +/- 128.79	31.22 +/- 15.27	91.96 +/- 149.05	70.08 +/- 107.84	0.51 +/- 0.08	70.56 +/- 2.01
e5-b512	697.39 +/- 127.73	31.26 +/- 14.83	89.74 +/- 144.29	67.99 +/- 104.73	0.60 +/- 0.03	72.63 +/- 2.25
e5-b1024	693.84 +/- 124.91	31.43 +/- 14.53	87.59 +/- 139.71	66.70 +/- 101.20	0.58 +/- 0.02	78.02 +/- 3.67
e10-b128	731.22 +/- 113.73	36.70 +/- 13.54	56.94 +/- 115.01	43.50 +/- 83.86	0.75 +/- 0.08	144.61 +/- 8.10
e10-b256	738.19 +/- 119.51	36.71 +/- 13.01	54.46 +/- 108.56	41.87 +/- 79.06	0.53 +/- 0.02	141.20 +/- 7.40
e10-b512	736.02 +/- 117.87	36.31 +/- 12.23	52.30 +/- 103.01	39.99 +/- 75.20	0.62 +/- 0.10	137.99 +/- 6.98
e10-b1024	733.21 +/- 116.55	36.74 +/- 12.13	50.34 +/- 98.04	39.32 +/- 71.14	0.63 +/- 0.04	145.35 +/- 8.55

Tabela 15: TF Encrypted no ambiente de execução de 4 GB e com recurso ao dataset MNIST half

test	used	usr	read	writ	acc	time
e1-b128	582.18 +/- 11.46	14.22 +/- 1.28	125.73 +/- 40.94	93.63 +/- 29.65	0.57 +/- 0.06	10.27 +/- 1.18
e1-b256	581.97 +/- 11.37	14.28 +/- 1.29	125.50 +/- 40.81	93.49 +/- 29.53	0.67 +/- 0.05	10.26 +/- 0.11
e1-b512	582.22 +/- 11.31	14.34 +/- 1.30	125.27 +/- 40.66	93.37 +/- 29.45	0.57 +/- 0.04	12.55 +/- 0.19
e1-b1024	582.09 +/- 11.69	14.43 +/- 1.31	125.00 +/- 40.47	93.18 +/- 29.31	0.60 +/- 0.05	17.13 +/- 0.45
e5-b128	581.96 +/- 11.75	14.57 +/- 1.34	124.64 +/- 40.24	92.93 +/- 29.14	0.66 +/- 0.05	39.45 +/- 3.07
e5-b256	582.16 +/- 11.52	14.76 +/- 1.38	123.84 +/- 39.65	92.36 +/- 28.71	0.56 +/- 0.08	40.12 +/- 3.32
e5-b512	581.97 +/- 11.57	14.95 +/- 1.41	123.06 +/- 39.16	91.81 +/- 28.37	0.56 +/- 0.05	39.98 +/- 0.92
e5-b1024	581.62 +/- 11.28	15.15 +/- 1.44	122.30 +/- 38.67	91.25 +/- 28.02	0.57 +/- 0.08	45.41 +/- 1.37
e10-b128	696.42 +/- 125.78	30.69 +/- 16.77	60.72 +/- 70.75	46.95 +/- 50.95	0.69 +/- 0.03	76.22 +/- 5.92
e10-b256	696.49 +/- 125.46	30.63 +/- 16.36	60.02 +/- 69.83	46.47 +/- 50.27	0.54 +/- 0.09	71.92 +/- 3.17
e10-b512	694.95 +/- 125.36	30.36 +/- 15.72	59.38 +/- 68.98	45.71 +/- 49.94	0.55 +/- 0.03	72.74 +/- 3.06
e10-b1024	693.37 +/- 123.31	30.37 +/- 15.41	58.74 +/- 68.14	45.15 +/- 49.44	0.63 +/- 0.07	80.40 +/- 10.60

Tabela 16: TF Encrypted no ambiente de execução de 8 GB e com recurso ao dataset MNIST full

test	used	usr	read	writ	acc	time
e1-b128	1353.23 +/- 2.26	68.87 +/- 1.24	78.76 +/- 2.75	176.24 +/- 8.83	0.51 +/- 0.10	15.89 +/- 0.13
e1-b256	1358.34 +/- 2.67	68.89 +/- 1.24	79.66 +/- 4.12	176.07 +/- 8.81	0.61 +/- 0.01	18.79 +/- 1.74
e1-b512	1357.51 +/- 2.80	68.92 +/- 1.24	79.57 +/- 4.12	175.87 +/- 8.80	0.56 +/- 0.03	22.13 +/- 1.91
e1-b1024	1357.40 +/- 2.44	68.96 +/- 1.23	79.46 +/- 4.10	175.63 +/- 8.77	0.64 +/- 0.05	29.04 +/- 1.59
e5-b128	1458.01 +/- 109.07	81.74 +/- 13.98	39.66 +/- 43.52	90.63 +/- 92.95	0.54 +/- 0.06	63.66 +/- 1.21
e5-b256	1468.24 +/- 121.73	81.70 +/- 13.83	39.52 +/- 43.36	89.70 +/- 93.30	0.55 +/- 0.03	65.65 +/- 3.89
e5-b512	1495.40 +/- 154.39	81.66 +/- 13.68	39.37 +/- 43.20	90.02 +/- 92.25	0.60 +/- 0.06	68.53 +/- 3.75
e5-b1024	1558.84 +/- 218.97	81.82 +/- 13.75	39.21 +/- 43.03	89.02 +/- 92.62	0.55 +/- 0.04	73.67 +/- 2.25
e10-b128	1495.23 +/- 102.40	86.17 +/- 12.60	26.03 +/- 39.10	60.25 +/- 84.47	0.73 +/- 0.08	124.22 +/- 1.80
e10-b256	1508.90 +/- 113.46	86.05 +/- 12.38	25.85 +/- 38.83	60.05 +/- 83.75	0.55 +/- 0.07	126.31 +/- 4.77
e10-b512	1508.62 +/- 117.68	85.96 +/- 12.18	25.67 +/- 38.56	60.62 +/- 82.44	0.63 +/- 0.02	129.09 +/- 4.53
e10-b1024	1524.68 +/- 139.45	86.02 +/- 12.11	25.49 +/- 38.28	61.73 +/- 80.77	0.58 +/- 0.14	135.51 +/- 4.93

Tabela 17: TF Encrypted no ambiente de execução de 8 GB e com recurso ao dataset MNIST half

test	used	usr	read	writ	acc	time
e1-b128	1347.62 +/- 2.81	84.53 +/- 0.26	110.78 +/- 4.80	251.94 +/- 9.61	0.61 +/- 0.06	10.17 +/- 0.13
e1-b256	1349.12 +/- 1.65	84.54 +/- 0.26	110.67 +/- 4.79	251.70 +/- 9.59	0.64 +/- 0.02	11.68 +/- 0.09
e1-b512	1349.71 +/- 1.75	84.55 +/- 0.26	110.55 +/- 4.78	252.12 +/- 10.61	0.56 +/- 0.15	15.31 +/- 0.12
e1-b1024	1349.38 +/- 1.85	84.56 +/- 0.26	110.39 +/- 4.77	251.78 +/- 10.59	0.62 +/- 0.13	22.70 +/- 0.09
e5-b128	1350.25 +/- 0.79	84.58 +/- 0.26	110.17 +/- 4.75	251.29 +/- 10.55	0.67 +/- 0.02	35.60 +/- 2.23
e5-b256	1350.41 +/- 1.00	84.61 +/- 0.25	109.84 +/- 4.72	250.56 +/- 10.49	0.64 +/- 0.04	34.84 +/- 0.22
e5-b512	1349.51 +/- 1.81	84.64 +/- 0.25	109.51 +/- 4.69	249.84 +/- 10.43	0.52 +/- 0.04	38.06 +/- 0.49
e5-b1024	1349.72 +/- 1.87	84.67 +/- 0.25	109.16 +/- 4.66	249.05 +/- 10.36	0.63 +/- 0.05	45.03 +/- 0.45
e10-b128	1449.74 +/- 110.60	89.59 +/- 5.35	54.38 +/- 59.64	126.81 +/- 133.08	0.71 +/- 0.04	65.14 +/- 2.07
e10-b256	1458.05 +/- 118.87	89.53 +/- 5.23	54.09 +/- 59.32	125.64 +/- 132.95	0.59 +/- 0.04	64.70 +/- 1.10
e10-b512	1474.74 +/- 140.43	89.57 +/- 5.22	53.80 +/- 59.01	125.14 +/- 132.10	0.61 +/- 0.06	66.15 +/- 0.67
e10-b1024	1506.05 +/- 181.24	89.49 +/- 5.07	53.51 +/- 58.69	124.68 +/- 131.19	0.70 +/- 0.04	73.73 +/- 1.64

Tabela 18: Modelo base no ambiente de execução de DI-4 GB e com recurso ao dataset MNIST full

test	used	usr	read	writ	acc	time
e1-b128	480.74 +/- 7.74	1.45 +/- 0.02	0.42 +/- 0.00	2.71 +/- 0.00	0.96 +/- 0.00	19.23 +/- 0.37
e1-b256	481.52 +/- 7.80	1.45 +/- 0.02	0.42 +/- 0.00	2.71 +/- 0.00	0.95 +/- 0.00	13.51 +/- 0.55
e1-b512	481.57 +/- 7.62	1.45 +/- 0.02	0.42 +/- 0.00	2.71 +/- 0.00	0.94 +/- 0.00	11.76 +/- 0.45
e1-b1024	481.37 +/- 7.91	1.45 +/- 0.02	0.42 +/- 0.00	2.71 +/- 0.00	0.93 +/- 0.00	10.78 +/- 0.62
e5-b128	780.16 +/- 327.16	33.64 +/- 35.26	0.21 +/- 0.23	4.42 +/- 1.87	0.98 +/- 0.00	71.38 +/- 2.64
e5-b256	632.07 +/- 300.66	18.44 +/- 33.98	0.31 +/- 0.21	2.86 +/- 0.29	0.98 +/- 0.00	53.26 +/- 2.49
e5-b512	481.82 +/- 7.60	1.46 +/- 0.02	0.42 +/- 0.00	2.71 +/- 0.00	0.97 +/- 0.00	46.49 +/- 1.32
e5-b1024	481.71 +/- 7.80	1.46 +/- 0.02	0.42 +/- 0.00	2.71 +/- 0.00	0.97 +/- 0.00	42.21 +/- 0.95
e10-b128	880.88 +/- 299.29	44.65 +/- 32.41	0.52 +/- 1.10	6.33 +/- 7.99	0.98 +/- 0.00	146.81 +/- 2.58
e10-b256	783.94 +/- 328.74	35.86 +/- 37.68	0.21 +/- 0.23	3.39 +/- 0.74	0.98 +/- 0.00	106.95 +/- 2.27
e10-b512	788.82 +/- 334.08	38.15 +/- 40.19	0.21 +/- 0.23	4.17 +/- 2.27	0.98 +/- 0.00	86.04 +/- 2.83
e10-b1024	796.96 +/- 342.92	39.62 +/- 41.79	0.21 +/- 0.23	3.10 +/- 0.68	0.98 +/- 0.00	75.54 +/- 2.16

Tabela 19: Modelo base no ambiente de execução de DI-4 GB e com recurso ao dataset MNIST half

test	used	usr	read	writ	acc	time
e1-b128	467.96 +/- 7.28	1.41 +/- 0.01	0.41 +/- 0.01	2.71 +/- 0.00	0.94 +/- 0.00	11.54 +/- 1.52
e1-b256	463.14 +/- 3.85	1.42 +/- 0.01	0.42 +/- 0.00	2.71 +/- 0.00	0.93 +/- 0.00	8.53 +/- 0.18
e1-b512	463.40 +/- 3.65	1.42 +/- 0.01	0.42 +/- 0.00	2.71 +/- 0.00	0.92 +/- 0.00	7.38 +/- 0.26
e1-b1024	463.39 +/- 3.60	1.42 +/- 0.01	0.42 +/- 0.00	2.71 +/- 0.00	0.90 +/- 0.00	7.11 +/- 0.09
e5-b128	462.85 +/- 4.37	1.42 +/- 0.01	0.42 +/- 0.00	2.71 +/- 0.00	0.97 +/- 0.00	38.49 +/- 0.69
e5-b256	462.80 +/- 4.35	1.42 +/- 0.01	0.42 +/- 0.00	2.71 +/- 0.00	0.97 +/- 0.00	28.63 +/- 0.44
e5-b512	463.06 +/- 4.21	1.42 +/- 0.01	0.42 +/- 0.00	2.71 +/- 0.00	0.96 +/- 0.00	25.11 +/- 0.51
e5-b1024	462.92 +/- 4.17	1.42 +/- 0.01	0.42 +/- 0.00	2.71 +/- 0.00	0.95 +/- 0.00	22.30 +/- 0.79
e10-b128	761.08 +/- 326.80	33.37 +/- 35.01	0.21 +/- 0.23	4.04 +/- 1.50	0.98 +/- 0.00	75.12 +/- 4.38
e10-b256	612.95 +/- 299.72	18.48 +/- 34.12	0.32 +/- 0.21	2.79 +/- 0.15	0.97 +/- 0.00	53.92 +/- 1.95
e10-b512	463.21 +/- 4.36	1.42 +/- 0.01	0.42 +/- 0.00	2.71 +/- 0.00	0.97 +/- 0.00	44.87 +/- 1.59
e10-b1024	463.23 +/- 4.42	1.42 +/- 0.01	0.42 +/- 0.00	2.71 +/- 0.00	0.97 +/- 0.00	39.82 +/- 0.70

Tabela 20: Modelo base no ambiente de execução de 4 GB e com recurso ao dataset MNIST full

test	used	usr	read	writ	acc	time
e1-b128	558.95 +/- 410.52	16.19 +/- 24.28	65.83 +/- 3.18	217.03 +/- 366.79	0.96 +/- 0.00	5.32 +/- 0.36
e1-b256	559.63 +/- 411.53	16.22 +/- 24.26	65.78 +/- 3.20	216.96 +/- 366.67	0.95 +/- 0.00	3.90 +/- 0.25
e1-b512	559.20 +/- 410.98	16.25 +/- 24.24	65.75 +/- 3.21	216.95 +/- 366.54	0.94 +/- 0.00	3.14 +/- 0.13
e1-b1024	559.23 +/- 410.63	16.27 +/- 24.22	65.72 +/- 3.22	216.90 +/- 366.46	0.93 +/- 0.00	2.84 +/- 0.19
e5-b128	559.11 +/- 410.95	16.29 +/- 24.21	65.69 +/- 3.23	216.85 +/- 366.38	0.98 +/- 0.00	19.97 +/- 1.87
e5-b256	559.36 +/- 410.79	16.39 +/- 24.13	65.56 +/- 3.28	216.66 +/- 366.05	0.98 +/- 0.00	14.04 +/- 1.01
e5-b512	559.30 +/- 410.60	16.47 +/- 24.08	65.47 +/- 3.31	216.49 +/- 365.76	0.97 +/- 0.00	10.64 +/- 0.74
e5-b1024	558.73 +/- 410.92	16.53 +/- 24.04	65.40 +/- 3.33	216.35 +/- 365.53	0.97 +/- 0.00	8.92 +/- 0.78
e10-b128	558.99 +/- 410.85	16.59 +/- 24.00	65.34 +/- 3.36	216.25 +/- 365.36	0.98 +/- 0.00	38.61 +/- 5.46
e10-b256	559.08 +/- 410.92	16.78 +/- 23.86	65.10 +/- 3.46	215.90 +/- 364.72	0.98 +/- 0.00	26.39 +/- 1.13
e10-b512	559.15 +/- 410.83	16.91 +/- 23.77	64.94 +/- 3.52	215.62 +/- 364.26	0.98 +/- 0.00	19.28 +/- 1.31
e10-b1024	559.06 +/- 410.44	17.03 +/- 23.69	64.82 +/- 3.56	215.43 +/- 363.93	0.98 +/- 0.00	15.55 +/- 0.86

Tabela 21: Modelo base no ambiente de execução de 4 GB e com recurso ao dataset MNIST half

test	used	usr	read	writ	acc	time
e1-b128	496.94 +/- 411.50	15.55 +/- 24.95	166.54 +/- 164.15	227.15 +/- 363.67	0.94 +/- 0.00	3.61 +/- 0.61
e1-b256	495.28 +/- 384.85	15.79 +/- 24.73	479.98 +/- 705.56	227.15 +/- 363.32	0.93 +/- 0.00	3.09 +/- 0.78
e1-b512	526.47 +/- 439.45	15.97 +/- 24.58	475.36 +/- 697.61	227.06 +/- 363.41	0.92 +/- 0.00	2.30 +/- 0.15
e1-b1024	526.37 +/- 439.77	16.14 +/- 24.44	471.11 +/- 690.30	226.90 +/- 363.38	0.90 +/- 0.00	2.04 +/- 0.07
e5-b128	526.31 +/- 439.86	16.30 +/- 24.32	467.00 +/- 683.22	226.76 +/- 363.39	0.97 +/- 0.00	9.57 +/- 0.31
e5-b256	526.49 +/- 439.80	16.80 +/- 23.93	456.31 +/- 664.84	232.95 +/- 358.15	0.97 +/- 0.00	6.80 +/- 0.08
e5-b512	526.74 +/- 439.14	17.17 +/- 23.66	448.67 +/- 651.71	232.58 +/- 358.21	0.96 +/- 0.00	5.27 +/- 0.10
e5-b1024	526.88 +/- 439.41	17.47 +/- 23.45	442.52 +/- 641.13	232.28 +/- 358.26	0.95 +/- 0.00	4.54 +/- 0.12
e10-b128	526.91 +/- 439.45	17.75 +/- 23.26	437.10 +/- 631.82	232.01 +/- 358.30	0.98 +/- 0.00	17.51 +/- 0.43
e10-b256	526.39 +/- 439.06	18.51 +/- 22.78	421.97 +/- 605.81	231.34 +/- 358.33	0.98 +/- 0.00	12.93 +/- 0.58
e10-b512	526.57 +/- 439.42	19.08 +/- 22.45	410.69 +/- 586.44	230.86 +/- 358.33	0.97 +/- 0.00	9.55 +/- 0.26
e10-b1024	526.46 +/- 440.29	19.51 +/- 22.23	402.66 +/- 572.65	230.42 +/- 358.37	0.97 +/- 0.00	8.33 +/- 0.93

Tabela 22: Modelo base no ambiente de execução de 8 GB e com recurso ao dataset MNIST full

test	used	usr	read	writ	acc	time
e1-b128	999.05 +/- 9.66	40.91 +/- 13.84	910.53 +/- 137.59	30.87 +/- 5.70	0.96 +/- 0.00	7.17 +/- 0.17
e1-b256	1000.14 +/- 9.58	41.35 +/- 13.71	902.36 +/- 134.93	30.65 +/- 5.68	0.95 +/- 0.00	5.74 +/- 0.04
e1-b512	1002.29 +/- 8.19	41.70 +/- 13.61	895.71 +/- 132.93	30.45 +/- 5.66	0.94 +/- 0.00	4.82 +/- 0.08
e1-b1024	1002.18 +/- 9.56	42.02 +/- 13.53	889.77 +/- 131.00	30.38 +/- 5.56	0.93 +/- 0.00	4.41 +/- 0.07
e5-b128	1002.02 +/- 8.84	42.30 +/- 13.45	884.26 +/- 129.33	30.23 +/- 5.49	0.98 +/- 0.00	30.76 +/- 0.70
e5-b256	1002.20 +/- 10.40	43.75 +/- 13.04	859.15 +/- 122.11	29.64 +/- 5.31	0.98 +/- 0.00	22.58 +/- 0.13
e5-b512	1002.45 +/- 11.50	44.79 +/- 12.77	841.09 +/- 116.88	29.18 +/- 5.24	0.97 +/- 0.00	18.50 +/- 0.03
e5-b1024	1002.81 +/- 11.75	45.62 +/- 12.55	826.52 +/- 112.74	28.84 +/- 5.16	0.97 +/- 0.00	15.52 +/- 0.25
e10-b128	1246.89 +/- 329.59	65.36 +/- 27.52	489.26 +/- 453.20	20.30 +/- 12.00	0.98 +/- 0.00	59.00 +/- 0.86
e10-b256	1013.50 +/- 7.34	48.54 +/- 11.84	777.90 +/- 97.45	27.98 +/- 5.52	0.98 +/- 0.00	43.10 +/- 0.16
e10-b512	1015.32 +/- 7.10	50.06 +/- 11.46	752.51 +/- 90.62	28.32 +/- 6.99	0.98 +/- 0.00	36.41 +/- 0.89
e10-b1024	1020.05 +/- 14.68	51.28 +/- 11.13	734.74 +/- 85.28	28.11 +/- 7.43	0.98 +/- 0.00	29.53 +/- 0.25

Tabela 23: Modelo base no ambiente de execução de 8 GB e com recurso ao dataset MNIST half

test	used	usr	read	writ	acc	time
e1-b128	998.46 +/- 18.21	19.92 +/- 10.48	1222.56 +/- 171.00	38.20 +/- 4.20	0.94 +/- 0.00	4.39 +/- 0.10
e1-b256	1005.80 +/- 16.99	20.55 +/- 10.36	1487.79 +/- 520.70	38.08 +/- 3.94	0.93 +/- 0.00	4.25 +/- 0.84
e1-b512	1009.62 +/- 15.05	21.28 +/- 10.09	1469.11 +/- 504.44	43.36 +/- 7.04	0.92 +/- 0.00	3.28 +/- 0.13
e1-b1024	1013.19 +/- 17.65	21.89 +/- 9.91	1453.54 +/- 491.87	42.94 +/- 6.79	0.90 +/- 0.00	2.85 +/- 0.19
e5-b128	1012.60 +/- 17.11	22.40 +/- 9.81	1440.91 +/- 483.41	42.61 +/- 6.63	0.97 +/- 0.00	16.33 +/- 0.57
e5-b256	1013.65 +/- 18.62	24.25 +/- 9.43	1399.36 +/- 454.31	41.84 +/- 6.36	0.97 +/- 0.00	11.90 +/- 0.33
e5-b512	1009.87 +/- 18.00	25.56 +/- 9.24	1370.37 +/- 436.56	41.12 +/- 6.08	0.96 +/- 0.00	9.99 +/- 0.12
e5-b1024	1011.43 +/- 17.99	26.65 +/- 9.09	1345.91 +/- 421.42	40.60 +/- 5.84	0.95 +/- 0.00	8.52 +/- 0.04
e10-b128	1011.41 +/- 16.99	27.59 +/- 8.98	1325.00 +/- 408.64	40.07 +/- 5.63	0.98 +/- 0.00	30.39 +/- 0.12
e10-b256	1011.43 +/- 18.47	30.29 +/- 8.73	1267.29 +/- 374.25	38.71 +/- 5.05	0.98 +/- 0.00	22.57 +/- 0.24
e10-b512	1011.69 +/- 19.48	32.17 +/- 8.58	1226.79 +/- 351.05	37.70 +/- 4.79	0.97 +/- 0.00	18.41 +/- 0.20
e10-b1024	1012.74 +/- 19.08	33.66 +/- 8.47	1195.05 +/- 333.38	36.95 +/- 4.56	0.97 +/- 0.00	15.49 +/- 0.12

4.1 Observações

Enumeram-se, de seguida, um conjunto de observações mais gerais derivadas dos resultados apresentados nas tabelas anteriores.

Observação 1. Como seria de esperar, o ambiente de execução DI-4 GB apresenta claramente os piores resultados em termos de tempos de execução das frameworks. No entanto, os ambientes de execução 4 GB e 8 GB apresentam tempos de execução muito semelhantes, sendo que por vezes o ambiente 4 GB é o mais eficiente. Além disso, o ambiente de 8 GB é ambiente onde mais memória RAM e carga de CPU são consumidos.

Observação 2. A framework TensorFlow Privacy é a mais custosa em termos de recursos computacionais e tempos de execução, em especial destaque para o ambiente DI-4 GB que possui, em média, tempos de execução cerca de 7x superiores aos outros ambientes. Alguns dos testes efetuados a esta framework neste ambiente chegam a ultrapassar os 5000 segundos (cerca de 1 hora e meia de execução). No entanto, possui uma accuracy de cerca de 80%. Visto que esta framework possui exatamente o mesmo modelo que o modelo base desenvolvido em TensorFlow, verifica-se que em média o overhead de accuracy perdido foi de 17% (a accuracy traduzida pelo modelo base é de cerca de 97%).

Observação 3. As frameworks PySyft e TensorFlow Privacy apresentam os maiores valores de operações de leitura e escrita em disco. Além disso, o desvio padrão destes valores é bastante elevado, sendo muitas vezes superior à média. Isto indica uma grande distribuição dos valores obtidos destas duas métricas, o que sugere "picos" de escrita e de leitura durante a execução destas frameworks.

Observação 4. A framework TF Encrypted apresenta claramente os piores resultados em termos da métrica accuracy. Como o modelo desta framework é relativamente semelhante ao modelo base pode-se efetuar uma comparação

direta com este. Ao efetuar-se tal processo verifica-se que em média a perda de accuracy resulta em cerca de 40% face ao modelo base.

Observação 5. Em termos de tempo de execução a framework TF Encrypted é a mais leve, ultrapassando cerca de 2 a 3 vezes os tempos de execução do modelo base. Também apresenta a menor quantidade de recursos computacionais usada, com a exceção do modelo base.

Observação 6. Os melhores resultados são geralmente apresentados para 10 épocas aplicadas a um batch size de 128. Além disso, os resultados para 10 épocas tendem a ser melhores para os resultados para 1 e 5 épocas, indicando que os modelos desenvolvidos para muitas frameworks possivelmente apresentariam melhores resultados para um valor ainda mais elevado de épocas.

Observação 7. Os resultados obtidos para o dataset MNIST half apresentam valores de accuracy bastante semelhantes quando comparados com a totalidade do dataset. Tal como esperado, os tempos de execução de metade do dataset são aproximadamente metade dos tempos de execução para o dataset completo. No entanto, a quantidade de recursos computacionais gasta é praticamente idêntica para os dois datasets.

Observação 8. A framework PySyft apresenta a melhor accuracy, com a exceção do modelo base. No entanto é difícil comparar este valor com o modelo base, já que o modelo desta framework é bastante mais complexo.

4.2 Análise dos Resultados

Tendo em conta os resultados apresentados nas tabelas acima, e as observações gerais descritas, efetuam-se, nesta subsecção, análises mais detalhadas dos resultados obtidos para cada framework. A análise foi efetuada segundo as características e algoritmos de cada framework, e com o uso da documentação fornecida pelos tutoriais usados. Como base de comparação tem-se o modelo criado em TensorFlow sem qualquer uso de framework de privacidade externa.

TensorFlow Privacy. Esta framework caracteriza-se como sendo a que gasta mais recursos computacionais e que apresenta o pior nível de performance. O pior caso acontece claramente para o ambiente DI-4 GB onde o tempo de execução ultrapassa os 7000 segundos. Esse mesmo teste efetuado com o modelo base do TensorFlow apenas demora cerca de 146 segundos no mesmo ambiente.

Muito provavelmente, os maiores valores em termos de tempos de execução e de recursos computacionais devem-se à implementação complexa desta framework. Como foi descrito na secção 2.2.1, esta framework utiliza uma implementação bastante complexa que assenta num conjunto de vários teacher models que introduzem ruído nas suas respostas. Esta noção claramente induz algum overhead. Além disso são efetuadas operações sobre dados com ruído

(o que muitas frameworks estudadas mencionam como sendo ineficiente) por parte de um agregador. Além destes modelos existe um outro modelo chamado student model. A introdução de vários modelos distintos pode resultar numa performance muito baixa e no aumento de recursos computacionais.

Depois de uma leitura da documentação relativa ao tutorial executado desta framework, verificou-se que a privacidade era estimada a partir de vários parâmetros, sendo que um desses parâmetros chamado epsilon mede a qualidade da privacidade obtida até um determinado momento. De acordo com o tutorial, este parâmetro é estimado para uma gama de valores de modo a se atingir um nível de privacidade desejado. Além disso, são necessárias operações que verificam se o nível de privacidade atingido, o que pode ser bastante custoso.

Após a execução desta framework, também se verificou que é gerada uma vasta quantidade de logs e checkpoints com diversas informações da execução. Nos ficheiros txt produzidos pelos testes há a indicação da localização desses ficheiros de logs produzidos. Ao verificar-se a dimensão desses ficheiros, verificou-se que os checkpoints ultrapassavam os 100 KB, e que eram gerados vários checkpoints por execução. Estes fatores evidenciam o facto desta framework efetuar vários recursos a nível de escrita e leitura em disco e que acrescentam o tempo de execução.

Devido aos aspetos mencionados, e tendo como base os tempos de execução e de recursos computacionais usados, pode-se afirmar que esta framework não é indicada para máquinas locais com poucos recursos. No entanto, face ao seu comportamento de cálculo e verificação de privacidade, esta framework poderá ser a que apresenta um maior rigor na segurança dos dados, traduzindo uma accuracy bastante elevada, ultrapassando os 90% no melhor caso.

PySyft. Esta framework traduz os melhores resultados em termos de accuracy, com a exceção do modelo base do TensorFlow. É difícil avaliar este aspeto com precisão, já que o modelo desenvolvido para esta framework é mais complexo que os modelos desenvolvidos para as outras frameworks. No entanto, como esta framework assenta no mecanismo de FL, não opera sobre dados encriptados nem existe o conceito de introdução de ruído como as outras frameworks. Em vez disso esta framework opera sobre o dataset enviando updates mínimos para o servidor global. Como isto acontece, é esperada uma maior accuracy face às outras frameworks e aproximação desse valor ao valor obtido pelo mesmo modelo não aplicado a esta framework. Por outro lado, este tipo de algoritmos não introduz um tipo de privacidade muito rigorosa, já que os dados acabam por ser enviados para o servidor global, só que em muito menores quantidades.

Os recursos computacionais gastos por esta framework são bastante inferiores quando comparados à framework TensorFlow Privacy. Este facto deve derivar, novamente, da complexidade muito inferior da implementação desta framework. No entanto, a framework TF Encrypted apresenta uma maior performance apesar de possuir algoritmos bastante mais complexos. Segundo os desenvolvedores desta framework, ainda há bastante melhorias a ser efetuadas,

sendo que reconhecem que a performance ainda não foi um aspeto tido em consideração.

Relativamente aos tempos de execução, são bastante superiores aos tempos de execução do modelo base em TensorFlow. Tendo em conta que esta framework não está a aplicar os algoritmos desenvolvidos num contexto real, já que são utilizados VirtualWorkers para simular apenas dois clientes, estes tempos de execução podem aumentar exponencialmente numa situação realista para múltiplos clientes. Tal como foi mencionado na secção 2.2.2, os custos relativos às comunicações com o servidor global são os mais elevados. Este facto demonstra que realmente esta framework não está otimizada e madura o suficiente para ser aplicada em dispositivos mobile que dispõem, normalmente, de menos recursos computacionais.

TF Encrypted. Fora o modelo base, esta framework apresenta claramente os melhores tempos de execução e menores gastos computacionais. Isto prova que foi atingida uma boa performance que consiste num dos conceitos tidos como base no desenvolvimento desta framework descrito na secção 2.2.3. No entanto, apresenta os piores resultados em termos de accuracy.

Segundo a documentação do exemplo aplicado desta framework, foram utilizados duas componentes essenciais. Uma dessas componentes constrói o modelo inicial e que efetua o treino localmente e envia uma encriptação dos pesos desse modelo para 3 servidores. A segunda componente envia o seu input encriptado para os servidores que operam sobre esses dados encriptados e enviam o output para essa componente que efetua a sua desencriptação. Analisando estes processos, verifica-se que o exemplo traduz uma grande complexidade. No entanto, tal não se verifica nos tempos de execução e nos recursos computacionais gastos, demonstrando que a performance foi tida bastante em conta. A rapidez desta framework pode estar associada ao facto de tirar partido do máximo de características da plataforma TensorFlow e de primitivas de otimização. No entanto, deve-se considerar que este processo não foi realista já que os servidores envolvidos consistiam numa simulação. Num contexto real em que ocorrem custos de comunicação entre esses servidores espera-se muito menos performance e um maior número de recursos gastos.

Quanto à baixa accuracy atingida, foi descrito por vários estudos efetuados às frameworks (secção 2) que as operações efetuadas em dados encriptados garantem um overhead significativo em termos de accuracy. Além disso, o modelo desenvolvido para esta framework segue uma abordagem de difícil comparação, já que esta framework apresenta as suas próprias estruturas de dados. Assim, esta accuracy baixa pode ser derivada de um modelo bastante simples, já que no estudo efetuado pelos desenvolvedores foi garantida uma accuracy mais elevada para modelos mais complexos (secção 2.2.3).

Recursos computacionais no ambiente de execução DI-4 GB. Como foi descrito na secção 3.2, este ambiente de execução caracteriza-se como sendo o que contém a menor quantidade de recursos computacionais. No en-

tanto, com a exceção dos maiores tempos de execução, os recursos computacionais gastos neste ambiente foram os que apresentaram valores mais baixos. No entanto estes valores podem não corresponder completamente à realidade já que consistiam em máquinas alojadas num servidor. Assim, pode ter sido negado o uso de todos os cores do CPU e da totalidade de memória RAM disponível. De notar que as operações de escrita e de leitura em disco apresentam valores bastante baixos. Isto pode derivar do facto das máquinas relativas a este ambiente de execução apresentarem pouca quantidade de disco disponível (a maior parte delas superava os 90% de utilização), não conseguindo efetuar tais operações. Alguns testes efetuados para a framework TensorFlow Privacy, que consome bastantes recursos deste género, foi interrompida indicando que não conseguia efetuar os checkpoints por falta de memória em disco.

Variação do número de épocas e batch size. Como foi falado, seguiu-se um estudo em que se fez variar o valor do número de épocas e de batch size. No entanto o estudo tido em consideração ([28]) ilustra resultados bastante diferentes dos obtidos. Foi descrito que a accuracy aumentaria com o aumento do batch size. No entanto, os resultados obtidos indicam que o menor batch size considerado apresenta os melhores resultados. Isto também pode querer dizer que valores inferiores a 128 traduziriam piores resultados. Também é descrito que quanto maior fosse o valor de batch size, maior seria o tempo de execução. No entanto, esta informação mostrou-se sem efeito para os resultados obtidos. Considerou-se a hipótese de que para batch sizes de valores mais pequenos, como ocorre um maior número de iterações, implica uma maior quantidade de cálculos de privacidade e que seria devido a essa questão os maiores tempos de execução. No entanto, verifica-se que isto não acontece mesmo para o modelo base sem qualquer tipo de privacidade.

Relativamente ao número de épocas, verifica-se que a accuracy atingida pelas frameworks tende a aumentar quanto maior for o número de épocas. Como o modelo base consegue atingir accuracies bastante elevadas para apenas 5 épocas, isto demonstra que a introdução de cálculos de privacidade necessita de um maior número de épocas para conseguir chegar a melhores resultados. Para a framework TF Encrypted, que produziu os piores resultados, seria interessante verificar se ocorreriam melhorias com o aumento significativo do número de épocas.

Dataset MNIST half. Com a utilização deste dataset era esperado que os recursos computacionais diminuíssem consideravelmente, assim como os tempos de execução e, consequentemente, a accuracy, já que a quantidade de dados de treino era muito superior. No entanto, apenas foi obtido o resultado esperado relativamente aos tempos de execução, que diminuíram aproximadamente para metade.

Os recursos computacionais mantiveram-se muito semelhantes, com a exceção da carga de CPU que diminui com este dataset. Isto pode querer dizer que a partir de um determinado tamanho inferior a este dataset a quantidade

de RAM usada passa a atingir valores elevados.

Surpreendentemente, a accuracy atingida por este dataset manteve-se bastante semelhante ao dataset total. Isto aconteceu para cada framework estudada. Este aspeto pode ter sido atingido devido ao facto de se ter extraído a primeira metade do dataset original e, de acordo com a documentação, os últimos elementos do dataset original serem mais ambíguos [29].

5 Conclusões e Trabalho Futuro

Neste trabalho explorou-se um conjunto de frameworks e diversos algoritmos de privacidade, estabelecendo-se espírito crítico sobre estas componentes. Segundo a metodologia dos testes efetuados, verificou-se que as frameworks introduzem em geral um grande overhead em termos de tempos de execução e de recursos computacionais gastos face a implementações sem qualquer processo de proteção de dados.

A nível dos resultados obtidos, verificou-se que a framework TensorFlow Privacy requer muito mais recursos computacionais que as restantes frameworks, mas que apresenta uma privacidade mais rigorosa por uma estimação exaustiva de inúmeros parâmetros e verificação da mesma. Quanto ao uso de cada framework, este depende do contexto do problema em questão. Se o intuito do modelo desenvolvido for rapidez aliada a uma proteção de dados rigorosa, mas não são necessários valores de accuracy elevados, então a framework TF Encrypted responde adequadamente a estes requisitos. Caso se pretenda uma proteção de dados bastante rigorosa, se disponha de recursos com um enorme poder computacional e é requerida uma accuracy elevada, poderá ser adequado usar frameworks mais complexas como o TensorFlow Privacy.

Como trabalho futuro sugere-se uma comparação mais justa em que se efetua uma uniformização dos modelos aplicados a estas frameworks. Tal como foi dito, algumas frameworks como TF Encrypted possuem as suas próprias implementações de modelos, o que consiste numa enorme dificuldade de se efetuar uma comparação rigorosa. Além disso, seria útil verificar as frameworks TF Encrypted e PySyft num contexto real sem qualquer tipo de simulação de clientes ou de servidores externos, de forma a se verificar o overhead realmente introduzido pelos seus mecanismos de privacidade e o tráfego na rede. Seria também interessante comparar as frameworks coMind Federated Learning e PySyft num tipo de estudo que visava responder qual das implementações deste tipo de algoritmos seria a mais segura e adequada para aplicações mobile. Adicionalmente, propõe-se efetuar testes para estas frameworks com um maior número de épocas, já que se verificou que os resultados obtidos tendem a melhorar significativamente com o aumento deste parâmetro. Propõe-se um estudo mais aprofundado dos parâmetros intrínsecos às frameworks de modo a verificar qual o seu potencial máximo. Para isso, sugere-se a utilização de datasets de menores dimensões, de modo a tornar este tipo de estudo exaustivo viável. Finalmente, sugere-se a simulação de diversos tipos de ataques que visam verificar a privacidade fornecida por cada framework.

Referências

[1] Shalev-Shwartz, Shai, Ben-David, Shai. (2014). Understanding Machine Learning: From Theory to Algorithms. In: Cambridge University Press

[2] LeCun Y., Bengio Y., Hinton G., (2015) Deep learning. In: Nature 521, 436-444

[3] `https://medium.com/georgian-impact-blog/a-brief-introduction-to-differential-privacy-eacf8722283b0`

[4] Papernot N., Song S., Mironov I., Raghunathan A., Talwar K., Erlingsson Ú. (2018) Scalable Private Learning with PATE. In: Conference at ICLR

[5] Papernot N., Abadi M., Erlingsson Ú, Goodfellow I., Talwar K. (2017) Semi-supervised knowledge transfer for deep learning from private training data. In: Proceedings of the 5th International Conference on Learning Representations (ICLR)

[6] Brendan McMahan H., Moore E., Ramage D., Hampson S., Agüera y Arcas B. (2017) Communication-Efficient Learning of Deep Networks from Decentralized Data. In: Google, Inc., 651 N 34th St., Seattle, WA 98103 USA

[7] Archer D., Bogdanov D., Kamm L., Lindell Y., Nielsen K, Pagter J. I., Smart N. P., Wright R. N. (2018) From Keys to Databases - Real-World Applications of Secure Multi-Party Computation

[8] Torres, D. (2015) Secure Multiparty Computation Protocols. In: Universidade do Minho, Escola de Engenharia

[9] Dahl M., Mancuso J., Dupis Y., DeCoste B., Giraud M., Livingstone I., Patriquin J., Uhma G. (2018) Private Machine Learning in TensorFlow using Secure Computation. In: Dropout Labs

[10] Keller M., Pastro V., Rotaru D. (2017) Overdrive: Making SPDZ Great Again.

[11] Boemer F., Lao Y., Wierzynski C. (2018) nGraph-HE: A Graph Compliler for Deep Learning on Homomorphically Encrypted Data

[12] Tramèr F., Boneh D. (2019) Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware. In: Conference at ICLR

[13] Ryffel T., Trask A., Dahl M., Wagner B., Mancuso J., Rueckert D., Passerat-Palmbach J. (2018) A generic framework for privacy preserving deep learning

[14] Cyphers, S., Bansal, A. K., Bhiwandiwalla, A., Bobba, J., Brookhart, M., Chakraborty, A., Constable, W., Convey, C., Cook, L., Kanawi, O., et al. (2018) Intel nGraph: An inter- mediate representation, compiler, and executor for deep learning

[15] Laine, K. (2018) Simple encrypted arithmetic library-SEAL (v2.3.1)

[16] Cheon, J. H., Kim, A., Kim, M., and Song, Y. (2017) Homomorphic encryption for arithmetic of approximate numbers

[17] Gentry, C. and Boneh, D.A. (2009) A fully homomorphic encryption scheme

[18] Halevi, S. and Shoup, V. (2014) Algorithms in HElib

[19] Chillotti, I., Gama, N., Georgieva, M., and Izabachene, M.Faster (2016) Fully homomorphic encryption: Bootstrapping in less than 0.1 seconds

[20] Stoica, I., Song, D., Popa, R.A., Patterson, D., Mahoney, M.W., Katz, R., Joseph, A.D., Jordan, M., Hellerstein, J.M., Gonzalez, J. E., et al. (2017) A Berkeley view of systems challenges for AI

[21] Ohrimenko, O., Schuster, F., Fournet, C., Mehta, A., Nowozin, S., Vaswani, K., and Costa, M. (2016) Oblivious multi-party machine learning on trusted processors

[22] Hunt, T., Song, C., Shokri, R., Shmatikov, V., and Witchel, E. (2018) Chiron: Privacy-preserving machine learning as a service.

[23] Cheng, R., Zhang, F., Kos, J., He, W., Hynes, N., Johnson, N., Juels, A., Miller, A., and Song, D. (2018) Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contract execution

[24] Motwani, R., and Raghavan, P. (2010) Randomized algorithms

[25] Simonyan, K., and Zisserman, A. (2014) Very deep convolutional networks for large-scale image recognition

[26] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017) Mobilenets: Efficient convolutional neural networks for mobile vision applications

[27] Abadi, M., Chu, A., Goodfellow, I., McMahan, H.B., Mironov, I., Talwar, K., and Zhang, L. (2016) Deep learning with differential privacy

[28] Radiuk M. P. (2017) Impact of Training Set Batch Size on the Performance of Convolutional Neural Networks for Diverse Datasets. In: Khmelnitsky National University, Ukraine

[29] `http://yann.lecun.com/exdb/mnist/`