小口研究室　研究紹介　(2022年度)
(お茶の水女子大学理学部情報科学科)

モバイル端末上でのプライバシに配慮した画像認識モデルを構築する手法の提案と実装
（研究担当:近藤 華）

研究背景

Androidなどのスマートフォンを始めとするモバイル端末は高機能化と高性能化が進んでいる

❖モバイル端末の高機能化
→ 深層学習技術を用いた画像認識技術が様々な問題を端末内で処理する

❖モバイル端末の高性能化
→ CPUのみならずGPU、NPUの搭載
→ メモリ、ストレージの大容量化

プライバシの保護のためモバイル端末に保存した画像データを外部に送信せずに、それらの画像データから得られた情報を用いた画像認識を行う手法が生まれた

❖転移学習技術を用いる手法
→ 演算にかかる時間は短いが精度があまり高くない

❖端末のデータを外部のサーバに送信し、演算させた結果を受け取る手法
→ 外部のサーバにデータを送信するため機密性の高い情報で学習を行いたい場合に適さない

モバイル(Android)端末内のみでファインチューニングを行いモデルの精度を高める、プライバシに配慮した画像認識モデルを構築する手法の提案

❖精度を高める
→ 転移学習よりも精度が高くなるであろうファインチューニングを使用
→ アーキテクチャには画像認識モデルとして精度が良いことが確認されているMobileNetを使用

❖ 個人情報になり得るデータを外部に送信しない
→ 端末内のみで端末内データを使用した学習処理を行うことで、学習に使用するデータのみならず学習後のモデルも端末外に出さない

実験概要

以下の２つの実験を行った。

① モバイル端末内のデータ（各種類100枚と少ないデータ量）とプロセッサのみが使用可能な環境下においてもファインチューニングを使用することで精度の高いモデルが作成可能であるかの検証実験
①の実験は、モバイル端末より実装が容易なPC上で行った

② モバイル端末内にモデルを組み込んだ場合にNNAPIを使用することで速度が向上するか確認を行う実験
学習しない層のみのベンチマークを測定した

実験結果

実験①
図1に矢印で示した層から出力層までの再学習を十分に行い、再学習後のモデルにおける精度を測定した。
結果は表1に示すようになった。

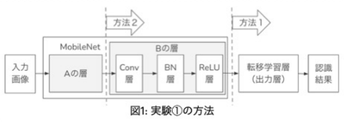


図1: 実験①の方法

表1: 実験①の結果

	方法 1	方法 2
精度 [%]	91.00	95.99

これにより、再学習用のデータが各種類100枚と少ない枚数でも方法2の手法である畳み込み層も再学習するファインチューニングを行う手法により精度の良いモデルを構築可能であることが確認できた。

実験②
環境は右の表2のものを使用した。
それぞれのベンチマークの結果は表3に示すようになった。
いずれの端末でも、NNAPI使用時の方が未使用時より実行速度が速くなることが確認できた。

表2: 実験②で使用した端末

Model number	Pixel4	Pixel5	POCO F2 Pro
OS	Android 12	Android 11	Android 10
SoC	Snapdragon 855	Snapdragon 765G	Snapdragon 865
Memory	6 GB	8 GB	8 GB

表3: ベンチマークの測定結果

Model number	Pixel4	Pixel5	POCO F2 Pro
NNAPI 未使用時	16203.4	16697.2	13664.8
NNAPI 使用時	5660.83	13826.7	6593.78

まとめ

モバイル（Android）端末向けのプライバシに配慮した精度の良い結果がえられる個人用の画像認識モデルを構築する手法として、モバイル端末上で畳み込み層も再学習するファインチューニングを行う手法を提案した。
また、モバイル端末上で実行可能なモデルの実装において、重みを固定する層ではAndroid Neural Networks API（NNAPI）を利用することで学習速度が向上することが確認できた。

今後の課題

Android端末上で実行可能なモデルの実装とアプリケーションの作成を行う。

❖モバイル端末上で実行可能なモデルの実装
→ 実験1の方法2の手法に従い、再学習を行う部分の実装も行う
→ 実装後に学習速度の計測を行い、実装方法やモデルの改善を行う

❖アプリケーションの作成
→ 実装したモデルがモバイル端末上で実用可能か調査するために作成する

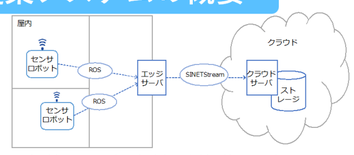
ROS準拠ロボット及びエッジを用いてストリーム処理を行うIoTシステムの構築と評価
（研究担当:佐々木　怜名）

背景

IoT機器から収集したセンサデータをスマートホームのサービスに活用
センサデータをクラウドに送信/蓄積/解析
(!) 室内全体に多数のセンサ設置は**コスト高**
(!) 大量のストリームデータをクラウドに集約し**処理量増加・通信遅延**
(!) **プライバシ・セキュリティ**の問題も考慮

研究目的
屋内を網羅的にセンシングするIoTシステムの構築

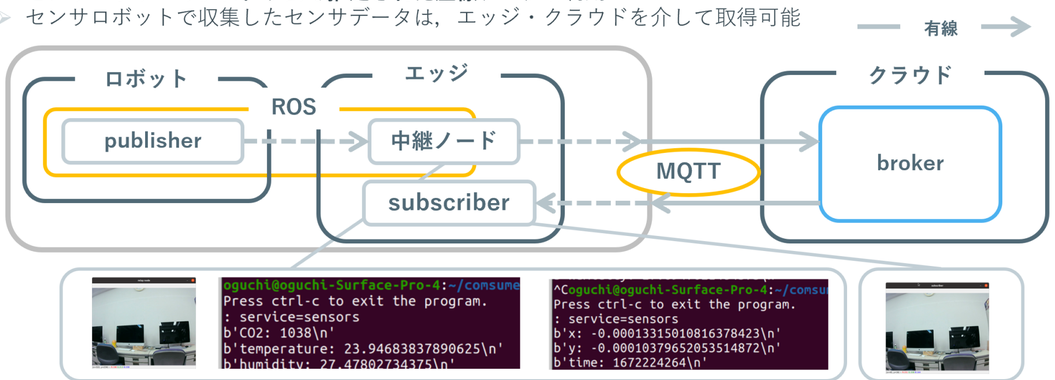
提案システムの概要



- ROSで実装された車輪型移動ロボットにセンサを搭載し，任意地点の室内環境情報収集
- センサロボットが収集したセンサデータをエッジサーバに集約，クラウドへ送信し解析処理

ストリーム処理に向けた転送実験

- センサロボットで収集した**カメラ画像/環境情報/位置情報**を中継ノード・brokerを介してsubscriberに転送
- センサロボットに**UVC(USB Video Class)カメラ**を搭載
- センサロボットに**SEK-SCD41(SENSIRION)**を搭載
- センサロボットの**プログラムで推定された座標データ**を利用
- センサロボットで収集したセンサデータは，エッジ・クラウドを介して取得可能

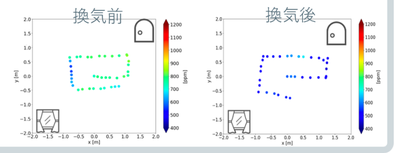


CO2濃度リアルタイム可視化アプリケーションの試作

- 室内でセンサロボットの遠隔操作を行いながらCO2濃度を収集し，リアルタイムで可視化するヒートマップを表示

publisher 環境情報と位置情報とタイムスタンプを収集
中継ノード 環境情報からCO2濃度を抽出
CO2濃度と位置情報をタイムスタンプで照合し，brokerにPub
subscriber brokerからSubした情報を元に， ヒートマップをリアルタイムで更新

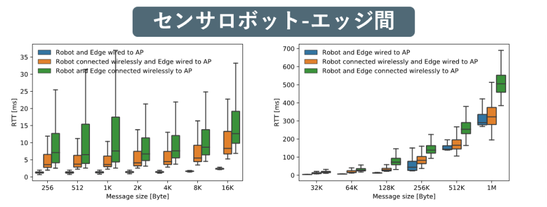
➢ **ロボットセンサを用いたIoTシステムの有用性を確認**

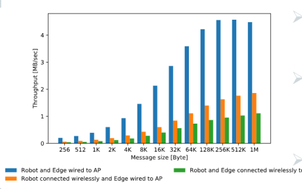


性能評価

異なるメッセージサイズ・異なる通信環境での転送実験による，遅延時間とスループットの評価

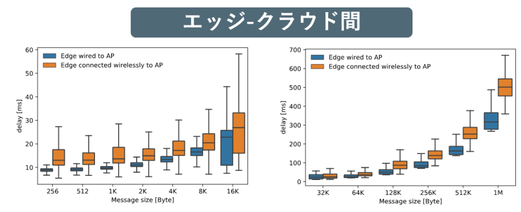
センサロボット-エッジ間

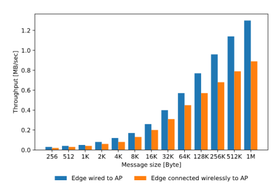




- 無線通信の方が遅延時間の分布/中央値が大きい
- 無線通信の時，データサイズが大きいときスループットが飽和
- データサイズが大きいとき，先行研究*より10倍の遅延時間

エッジ-クラウド間





- 無線通信の方が遅延時間の分布/中央値が大きい
- 無線通信の方がスループットが大きい

* Y. Maruyama, S. Kato and T. Azumi, "Exploring the performance of ROS2," *2016 International Conference on Embedded Software (EMSOFT)*, Pittsburgh, PA, USA, 2016, pp. 1-10, doi: 10.1145/2968478.2968502.

今後の課題

- **異なる転送レート**での転送実験/**ROSとROS2の通信を比較**した転送実験
- 効率的な環境情報収集に向けた，複数センサロボットの適切な制御方法を検討

ゲノム秘匿検索のbootstrap処理における適切な暗号ライブラリの検討
（研究担当:辻　有紗）

研究背景

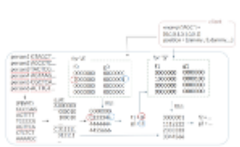
1 課題
・完全準同型暗号 (FHE)のHElibライブラリを用いたゲノム秘匿検索は低速
-> 実行時の負荷を計測し、思い処理の改善を検討
・FHEの時間・空間計算量が大きい
-> FHEのライブラリ・暗号方式を比較し、ハードウェアを用いた効率化やアプリケーションの処理内容に合わせた適切な暗号処理を考察

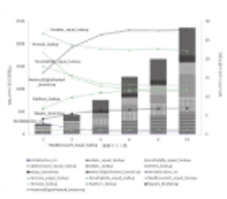
2-a ゲノム秘匿検索
クライアントサーバ 型通 信で, クライアントがサーバのゲノムデータベース (PBWT) に対して特定の塩基配列が最長マッチを持つか問い合わせを行う.

2-b ゲノム秘匿検索の手順
(1) クライアントはクエリを全文暗号化し, その他のパラ メータと一緒にサーバに送信する.
(2) サーバはクエリを元に FHE 演算を行い, 結果をクラ イアントに送信する.
(3) クライアントはサーバから送られてきたデータを復号 し, 結果を得る.

評価 –ゲノム秘匿検索の負荷

・35% Rotation 関数：暗号文同士の照合を行う。HElib では暗号文同士の比較ができないため Rotation の回数が多い
・30% bootstrap：bootstrap 処理内の extractDigitPacked 関数の負荷が大きい。extractDigitPacked は 1 回にかかる負荷が大きい


ゲノム秘匿検索の流れ

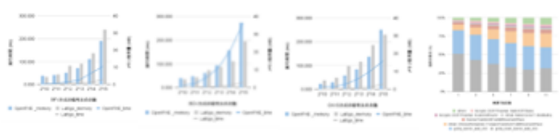

ゲノム秘匿検索の負荷の内訳

完全準同型暗号

1 特徴
・ 暗号化した状態で加算・乗算が可能な格子暗号
・整数の評価が可能
・暗号化したまま分析を行い、分析時の個人情報の漏洩を防ぐ
・bootstrap処理の時間空間計算量が大きい

2 暗号方式によるbit配置の違い
・BFV方式：C1 = [PK1・u + e1 + ΔM]q, C2 = [PK2・u + e2]q
・BGV方式：C1 = [PK1・u + t・e1 + M]q, C2 = [PK2・u + t・e2]q
・CKKS方式：C1 = [PK1・u + e1 + M]q, C2 = [PK2・u + e2]q

評価 – FHE暗号の乗算



・速度
– LattigoライブラリのCKKS方式が最速
– BGV, CKKS方式ではOpenFHEに比べLattigoが高速
– OpenFHEでは暗号方式の違いによる差は見られない
– LattigoではCKKS, BGV, BFVの順に高速
・負荷
– GOMPによるスレッド停止が60%以上を占める
– 乗算可能回数の増加に伴い、CRTパッキングで作成された多項式に対し乗算のためbit反転を行うための処理の負荷が増加
-> CRTパッキングの作成時に呼び出されるswitchformat関数と, CRTパッキングされた関数について暗号文を分解する CRTDecompose関数　に呼び出される

TFHE暗号

・1bitごとに論理回路(AND, OR, NANDなど)で評価し、ゲート毎bootstrapを行うことで高速処理が可能
・LUT(LookupTable)を用いて非線形関数の評価が可能
・実数の評価が可能

(Bootstrapの流れ)
1.Bootstrapping keyの作成
2. Blind Rotation
LUTに X−Δm+e を乗算することで多項式の係数を回転し、0次の係数にΔmのノイズが削減された暗号文を移動
3. Sample extract
4. Key switching

評価 – TFHE暗号の乗算

・速度
– OpenFHEのbootstrap 21723msに対しTFHEppでは69.21msと高速
– メモリ使用量はTFHEppが3GB、OpenFHEが7.4GB
・処理の負荷
(OpenFHE)
– 47.61% ForwardTransformToBitReverseInPlace:
CRTパッキングされた多項式について乗算のためbit反転、Sample extract から呼び出される
– 23.93% ModMulEq:
ベクトルに対しスカラーの剰余乗算
(TFHEpp)
– 31.82% trgswfftExternalProduct:
Decompositionにより作成されるTRGSWとTRLWEの積を計算
– 17.43% IdentityKeySwitchWithEncoder:
bootstrapにより新たな秘密鍵S'を元に作成された暗号文を、bootstrap前の秘密鍵Sによる暗号文に変換

今後の課題

・ 負荷が重い関数についてSSDの適用を検討
・TFHEまたはFHEのCKKS方式を用いて実装を行いSSDに適切な暗号処理を検討

https://www.is.ocha.ac.jp/~oguchi_lab/　　oguchi@is.ocha.ac.jp