# IoT システムにおけるワークフローモデルを用いた AI 推論処理の割当て最適化

伊藤 弘毅
三菱電機株式会社
Ito.Hiroki@dr.MitsubishiElectric.co.jp

徳本 晋
富士通株式会社
tokumoto.susumu@fujitsu.com

栗田 太郎
ソニー株式会社
taro.kurita@sony.com

石川 冬樹
国立情報学研究所
f-ishikawa@nii.ac.jp

## 要旨

エッジデバイスの計算リソースの増大に伴い，AI の推論処理をクラウド上ではなくエッジ側で実行するエッジ AI を実現できるようになり，AI を IoT システムに搭載する際の設計の選択肢が広がっている．AI の推論処理を適切にクラウド側とエッジ側に割当てない場合，処理性能が想定を下回ったり，高額なクラウド課金が発生するなど，要求品質を満たさない可能性がある．決定した処理割当て方法が要求品質を満たすか検証する必要があるが，実機を用いて検証する場合，検証用ソフトウェアの開発やシステムの環境構築に工数を要する．本論文では，AI の推論処理および付随する前処理と後処理の実行時間と通信量を表現するワークフローモデルを構築することにより，処理割当てのパターンごとに全体の処理時間や課金額を推測し，最適なパターンを開発者に提示する手法を提案する．また，推測結果と実機の実測結果を比較することにより，提案手法の精度と制約について論ずる．

## 1. はじめに

クラウドとエッジデバイスを連携させた IoT システムを構築し，システムから収集したデータを活用したソリューションを創出することにより，顧客に新たな価値を提供する試みが継続して行われている．IoT の市場規模は 2021 年に前年比 22.4%増加しており，2027 年まで継続して拡大する予測[1]もあり，将来的にも企業のビジネスにとって

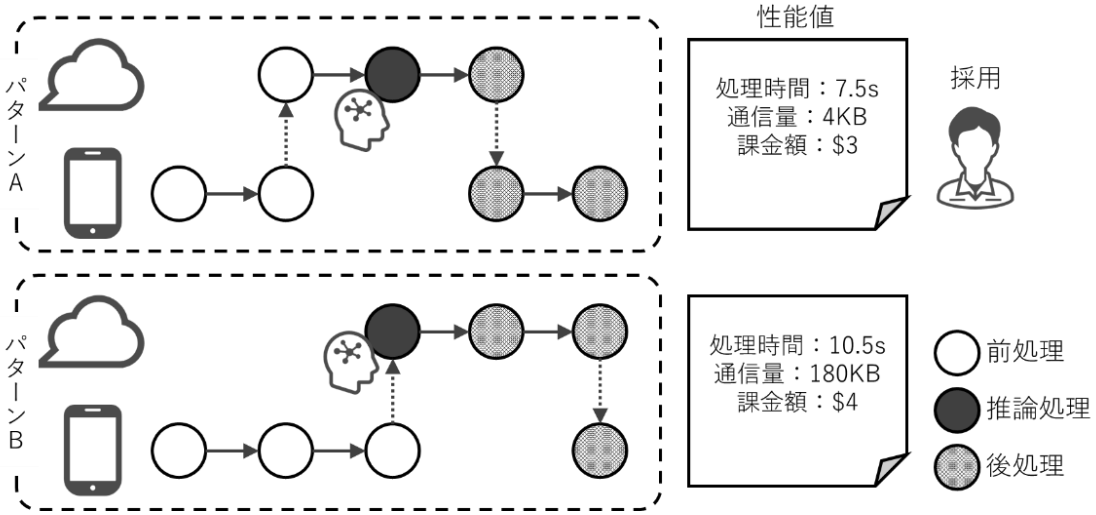



図 1 推論フローの処理割当ての決定

重要な要素となると思われる. IoT システムと AI は親和性が高く, エッジデバイスから収集したデータを AI に学習させることで, ユーザに価値を提供するソリューションを効率的に提供することが可能になる.

AI を搭載した機能を IoT システムにデプロイする場合, 比較的計算リソースの潤沢なクラウド上で推論処理を実行し, 実行結果をエッジデバイスに返す構成としていたことが多い. しかし, AI の推論処理をクラウドで実行する場合, エッジで収集したデータをクラウドに転送する必要が出てくるため, 通信帯域を多く消費するほか, クラウドのサーバに対する負荷も増大する[2]. そのため, 近年ではリアルタイム性能の確保やセキュリティの確保, クラウド利用による課金額の削減を目的として, エッジデバイス上で推論処理を実行するエッジ AI の構成を採用する場合も出てきている.

上記背景から, AI をシステムに搭載する時に, 開発者は推論処理と付随する処理をクラウドとエッジでどの処理を実行するか決定する必要がある. その際, 搭載機能に対する要件を考慮して処理の割当てを決定する必要があり, 開発者が複数ある候補の中から最適なものを選択する難易度は高い.

そこで, 本論文ではワークフローモデルと呼ばれる AI 推論フローを抽象化したモデルを構築することにより, 最適な処理割当て方法を導出する方法を提案する.

## 2. 課題

IoT システムに AI の推論フローを実装する場合, フローを構成する処理をクラウドまたはエッジのどちらに割当てる. 図 1 に, IoT システムに AI の推論フローをデプロイする時の, 処理の割当て例を示す.

AI の推論フローは, 学習モデルへ入力するデータを加工するための前処理と, 推論結果をアプリケーションが利用する形式に加工する後処理を含んでおり, これらの処理が逐次的に実行されることにより全体の機能が実現される. 推論フローをクラウドとエッジで連携して実現する場合, 処理の間でデータを送受信しながら処理を実行する.

AI の推論処理を含めた機能を搭載する場合, 要求品質を考慮して当該機能が実行する処理をエッジ側で実行するかクラウド側で実行するかを判断し実装する必要がある. 処理割当ての候補は, 図 1 に示すとおり複数考えられるが, 各処理の計算量や処理間の通信量の違いから, 候補によって処理時間やリソース使用量が変化する. そのため, 誤った候補を選択すると, エッジ―クラウド間の通信量が多くなったり, クラウドの処理により過大な課金額が発生したりする可能性がある.

実際の性能値や課金額は, 実機を用いて機能を搭載したシステムを稼働して検証する必要があるが, 機能実装完了後のシステム検証時において問題が明らかになると, 設計をやり直し再実装する必要が出てくるため, 大きな手戻りとなる. そのため, 機能実装前に処理時間などの性能を見積もる必要があるが, 数多くの処理割当ての組み合わせの中から, 要件を満たす最適な方法を見つけ出すのは困難である.

上記の課題に対して, 数理的なアルゴリズムを適用することにより, クラウドとエッジが連携して実現される機能について, 処理割当ての最適解の導出を試みた先行研究が複数存在する. Haghighi らは, 処理の流れをグラフで表現したモデルを構築し, 消費電力と処理時間を制約条件として費用を最小化するアルゴリズムを提案している[3]. Zhang らも, 処理フローをグラフで表現し, 消費電力を最小化するアルゴリズムを提案している[4]. また, 問題領域を数式化しマルコフ決定過程やディープラーニングを利用して, 同課題の解決を図る手法も提案されている[5][6].

しかし, 実開発への適用を考えた場合, 先行研究の手法で導出された最適解のみから処理割当てを決定することは難しい. 実開発ではステークホルダへの説明のため, 複数の処理割当て候補を比較し, 選択した候補がどの程度優位か説明を求められる場合もある. また, 推測された性能値が, 実システムで動作させた時の実測値と比較して許容できない誤差がないかを評価する必要がある. すなわち, 処理割当て方法を効率的に決定することに加えて, 結論の説明性と精度を考慮した手法を提案する必要がある.

本論文の研究課題を以下に示す.

**RQ1(説明性):処理時間やリソース利用量の推測結果から割当て方法を比較検討できるか**

**RQ2(精度):推測された処理時間やリソース利用量は実測値を精度良く再現できるか**

**RQ3(効率性):提案手法の推測は, 実機を用いた実測よりも効率良く候補を導出できるか**

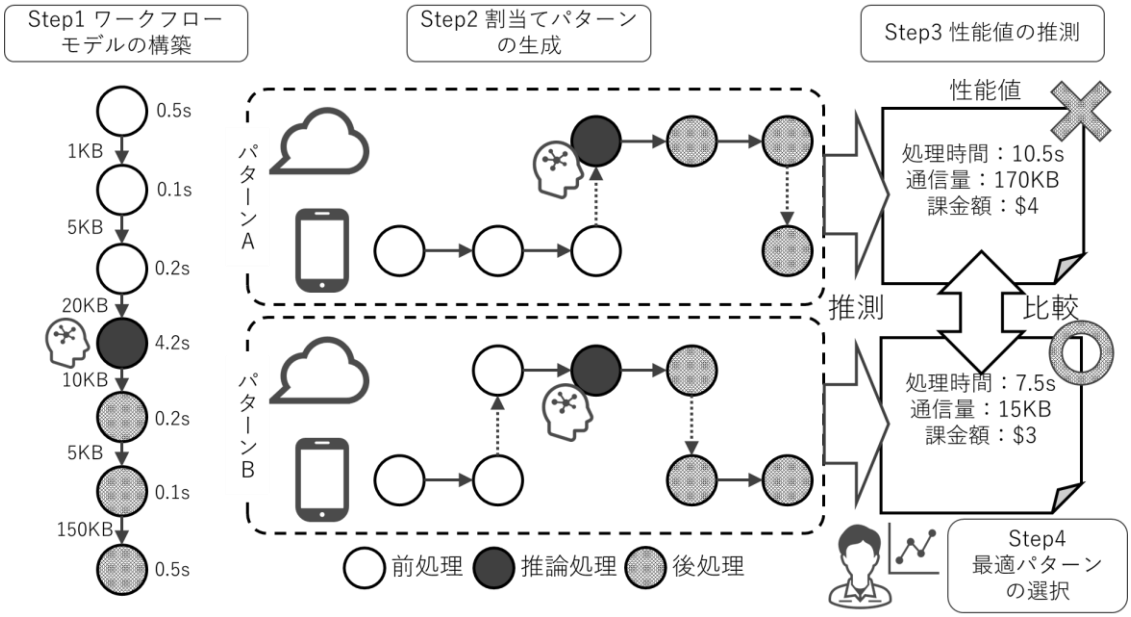



**図 2 提案手法の全体像**

## 3. 提案手法

AI の推論フローにおいて，処理の最適な割当てを導出する手法の全体像を図 2 に示す．提案手法は以下の 4 ステップから構成される．以下，詳細を説明する．

Step1 ワークフローモデルの構築
Step2 割当てパターンの生成
Step3 性能値の推測
Step4 最適パターンの選択

### 3.1. ワークフローモデルの構築

実装する推論フローを表現するワークフローモデルを構築する．ワークフローモデルは推論フローにおける処理をノード，使用データの流れをエッジとして表現される．ワークフローモデルのノードに対象処理の実行時間，エッジに処理間で渡されるデータのサイズを設定する．設定する値は，実機や汎用 PC で推論フローを実行した時の実測値とする．

### 3.2. 割当てパターンの生成

ワークフローモデルで表現した各処理について，実行する装置をクラウドまたはエッジに配分した割当てパターンを生成する．処理のうち実行装置を固定するものを決定し，それ以外の処理をクラウドまたはエッジに割当てる．処理割当てについては，想定される全ての組合せをパターンとして生成して性能値を推測する．

### 3.3. 性能値の推測

各割当てパターンについて，推論フローを実行した時の性能値を推測する．提案手法では，実機での処理時間とクラウドーエッジ間の通信量，クラウド課金額の 3 つを推測する．

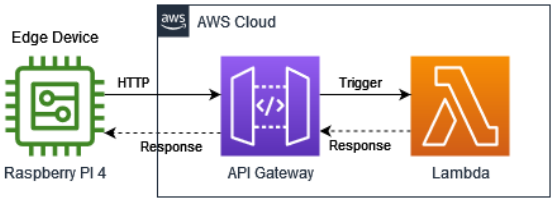
処理時間は，推論フローの処理時間を測定した実測マシンと実機マシンの性能比から推測する．推測処理時間は，以下の式から算出される．

$$推測処理時間 = 実測処理時間 \times \frac{実測マシン性能値}{実機マシン性能値}$$

マシンの性能値には，CPU のスペックを設定する．例えば，CPU のクロック周波数や PassMark[7]等のベンチマークの値が該当する．また，処理の特性に応じて性能値に追加の補正を加える．例えば，マルチスレッドにて動作する処理の場合，使用可能な CPU のコア数を性能値に乗算して補正する．また，ダウンロード処理やファイルの保存処理など CPU スペックに処理時間が依存しない処理に関しては，性能比による補正を実施しない．

通信量は，処理の実行装置が切替わるフローのデータサイズを加算して推測する．その際，課金額の算出式を考慮し，エッジからクラウドへの通信量とクラウドからエッジへの通信量を分けて算出する．

また，データの通信時間を以下の式で算出し，処理時




| | | |
|---|---|---|
| クラウド | サービス | AWS Lambda |
| | CPU | Intel Xeon Processor E5-2680 v3 |
| | クロック周波数 | 2.5GHz |
| | 割当てコア数 | 3 |
| | メモリ | 4096MB |
| エッジ | デバイス | Raspberry PI 4 Model B |
| | CPU | ARM Cortex-A72 |
| | コア数 | 4 |
| | クロック周波数 | 1.5GHz |
| | メモリ | 8GB |

図 3 本実験のシステム構成

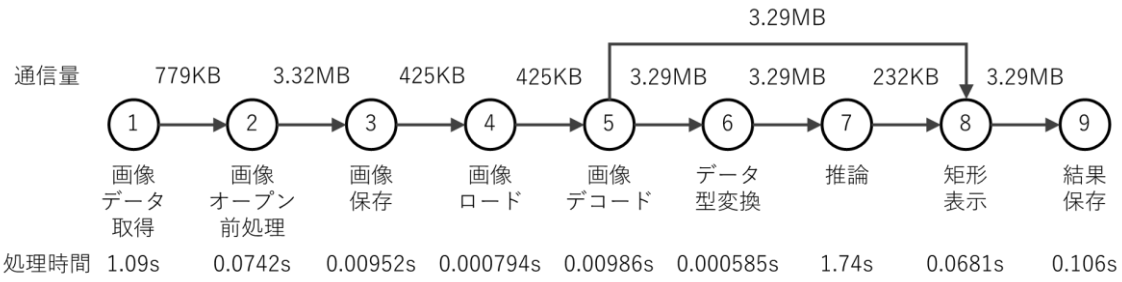



図 4 物体検知のワークフローモデル

間に加算する.

$$通信時間 = \frac{通信データサイズ}{通信速度}$$

課金額は，使用するクラウドサービスを想定し，当該サービスの料金体系をもとに導出する．例えば，データ転送の料金の場合，想定するユースケースをもとに単位時間当たりのリクエスト数を定数パラメータとして設定し，推測した通信量を入力して利用額を算出する．

### 3.4. 最適パターンの選択

各割当てパターンについて推測した性能値をもとに，開発者は最適な割当てパターンを選択する．一般的に，IoT システムにおける処理時間と課金額はトレードオフの関係にある．すなわち，CPU 性能の高いクラウドで処理を実行すると処理時間は短くなるが，クラウドとの通信量やサービス利用による課金額は増大する傾向にある．

よって，提案手法が対象とするのは最適解が一意に定まらない多目的最適化問題であり，複数の最適解の中から対象機能の要求品質を満たす最適パターンを選択する必要がある．選択に際し，複数の性能指標を評価軸として設定し，推測した性能値をプロットすることでパレートフロントを導出する．ここでパレートフロントは，プロットした割当てパターンのうち，他のどの解にも優越されない解(パレート解)を繋げたトレードオフ曲線である．導出したパレートフロントを確認し，対象機能の要求品質を満たす最適な割当てパターンを選択する．

## 4. 実験

### 4.1. 実験概要

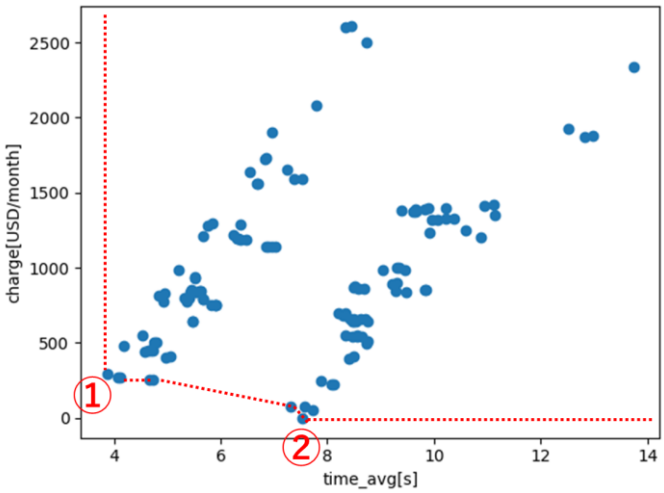
筆者は，物体検知の推論フローを表現したワークフローモデルを構築し，クラウドとエッジで構成されるシステムに実装した時の処理時間と通信量を推測する実験をした．

本実験のシステム構成を図 3 に示す．クラウドのプロバイダーは AWS とし，AWS Lambda を利用したサーバーレスな構成にて推論フローの処理を実行する想定とした．AWS Lambda の実行トリガーは API Gateway とし，外部から HTTP 通信で処理に必要なデータを受け取り処理を実施する．エッジ側は，デバイスは Raspberry PI 4 を使用することを想定し，一部処理をクラウド側に HTTP 通信でリクエストして実行することにより，全体の推論フローを実現する．

物体検知の推論は，フレームワークは TensorFlow を，学習済みモデルは COCO2017 のデータセットで学習済みの Faster R-CNN Resnet-50 V1[8]を利用した．

表 1 実験で使用した CPU の性能

| | 実測PC | クラウド | エッジ |
|---|---|---|---|
| CPU | Intel Core i7-1185G7 | Intel Xeon E5-2680 v3 | ARM Cortex-A72 |
| クロック周波数 | 3.0GHz | 2.5GHz | 1.5GHz |
| 総コア数 | 4 | 12 | 4 |
| 割当てコア数 | 2 | 3 | 4 |
| Integer Math | 35743 | 62261 | 5676 |
| Integer Math(1コア相当) | 8936 | 5188 | 1419 |




| | パターン① | パターン② |
|---|---|---|
| 処理割当て | クラウド：2～8<br>エッジ：1、9 | クラウド：なし<br>エッジ：1～9 |
| 処理時間(平均) | 3.88s | 7.54s |
| 課金額 | 297USD/month | 0USD/month |

図 5 割当てパターンの処理時間と課金額

### 4.2. 提案手法の適用

本実験で構築した物体検知の推論フローのワークフローモデルを図 4 に示す. 当該フローは 9 つの処理から構成され, 逐次的に処理が実行される. 処理 1 から 6 までが推論の前処理, 処理 7 が推論の本処理, 処理 8 と 9 が後処理に相当する. モデルの各処理に処理時間を, 処理間のフローに通信量を設定している. モデルに設定した処理時間と通信量は, 物体検知の各処理を汎用 PC 上で動作させた時の実測値を反映している.

上記ワークフローモデルの処理について, 各処理をクラウドとエッジに配分した割当てパターンを生成した. 本実験では, 処理 1 と 9 の割当てをエッジに固定し, それ以外の 7 つの処理をクラウドかエッジか選択して割当てるようにパターンを生成した. 割当てパターンの総数は, 処理 2 から 8 をクラウドまたはエッジに割当てた時の全組合せとなる 128(=$2^7$)である.

各割当てパターンの処理時間や通信量, 課金額の推定方法を説明する. 推論フローで実行される各処理は, 実行する CPU の性能により処理時間が変化する. ワークフローモデルに設定した処理時間は実測 PC 上で実行したときの値のため, クラウドまたはエッジ上での処理時間を CPU 性能に応じて補正する必要がある. 本実験では, 処理割合の大きい推論処理の整数演算のベンチマーク(Integer Math)値を利用し, クラウドとエッジで想定する CPU と実測 CPU のベンチマーク値の比を乗算し, 処理時間を補正した. 具体的な推測処理時間の算出式は以下の通り. ただし, 推論処理(処理 7)はマルチスレッドで動作するため, Integer Math(1 コア相当)の値にコア数を乗算した値で処理時間を計算する. 一方, ダウンロード処理やファイル保存処理(処理 1, 3, 9)には CPU 性能に関連する補正計算を実施しない.

$$推測処理時間 = 実測PC処理時間 \times \frac{実測CPU\ Integer\ Math(1\ コア相当)}{想定CPU\ Integer\ Math(1\ コア相当)}$$

測定に使用した実測 PC の CPU 性能と, 本実験で想定するクラウドとエッジの CPU 性能を表 1 に示す. 通信量は, 処理の間で実行装置が変わる場合に通信が発生

表 2 推測値と実測値(平均)の比較

| | | 総時間 | 処理1 | 処理2 | 処理3 | 処理4 | 処理5 | 処理6 | 処理7 | 処理8 | 処理9 | 通信時間 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| パターン① | 予測値 | 3.88 | 1.12 | 0.126 | 0.0097 | 0.00138 | 0.01731 | 0.000997 | 1.99 | 0.116 | 0.107 | 0.396 |
| | 実測値 | 6.38 | 0.991 | 0.0595 | 0.0070 | 0.00064 | 0.00724 | 0.000510 | 1.93 | 0.045 | 0.118 | 3.51 |
| パターン② | 予測値 | 7.54 | 1.08 | 0.466 | 0.0097 | 0.00509 | 0.0609 | 0.00350 | 5.39 | 0.420 | 0.101 | ー |
| | 実測値 | 13.96 | 0.906 | 0.122 | 0.0224 | 0.00167 | 0.0293 | 0.00173 | 12.5 | 0.224 | 0.117 | ー |

するものとし，推論フロー全体の通信量に加算する．クラウド―エッジ間のネットワーク速度は，100Mbps とした．課金額は，使用するクラウドサービスの課金の計算式に，推測した処理時間や通信量を代入して導出する．本実験では，5 秒に 1 回の割合で推論フローがリクエストされるものとし，一か月あたりの金額を算出した．

## 5. 実験結果

### 5.1. 最適パターンの導出

割当てパターンごとに処理時間と課金額の推測結果をプロットしたグラフを図 5 に示す．また，処理時間と課金額を評価軸としたときのパレートフロントを点線で示した．図 5 のパレートフロントを確認すると，処理時間が最小のパターン(図中①)と課金額が最小のパターン(図中②)が，開発者が選択する最適解として有力な候補となると考えられる．

上記 2 つの候補の概要を図 5 右側の表に整理した．パターン①は割当て可能な処理を全てクラウド側で実行し，パターン②はすべての処理をエッジ側で実行する．

### 5.2. 実測値との比較

筆者は，前節で導出した推定値の精度を確認するため，図 3 のシステムを実際に構築し，パターン①と②の推論フローを実際に動作させた．そして，各処理の実測処理時間と通信時間を測定して平均値を算出し，推定値と比較した．結果を表 2 に示す．パターン①と②の双方について，推測値と実測値の間に誤差が発生していることが分かる．以下に，発生誤差について考察する．

パターン①の通信時間については，実測の通信時間の方が長い結果となっている．この誤差は，提案手法の通信時間の予測値がデータの転送時間となっており，ネットワークの輻輳等を考慮していないため発生したと考えられる．ネットワーク輻輳の表現などモデルを改良することで，通信時間の予測精度が改善する可能性がある．

パターン②の処理 7 は，予測値よりも実測値の方が約 2.3 倍長い結果となった．処理 7 は推論処理となっておりマルチスレッドで動作する．マルチスレッドの動作によりエッジの CPU リソースを全て消費してしまい性能劣化した可能性があるが，他のモデルで動作させた時の動作と比較し，原因を検証する必要がある．

### 5.3. 考察

本論文の研究課題について，以下にて考察する．

**RQ1(説明性)：処理時間やリソース利用量の推測結果から割当て方法を比較検討できるか**

物体検知の推論フローを対象として，ワークフローモデルを構築することにより，割当てパターンごとの処理時間や通信量，課金額を机上で推測できることを確認した．また，割当てパターンごとに推測した処理時間や通信量，課金額を評価軸として設定し，パレートフロントを導出することにより，最適な処理割当てパターンの候補を複数導出し，比較検討できることを確認した．

本論文の検証に用いた推論フローは，構造が簡潔なものとなっている．処理が複雑なフローに対してワークフローモデルを構築する時に発生する課題があるか検証する必要がある．また，通信のスループットなどの制約条件をモデルに組み込み，想定する IoT システムを構築する上での前提を満たしているかについて，確認できる仕組みにした方が良いと考える．

**RQ2(精度)：推測された処理時間やリソース利用量は実測値を精度良く再現できるか**

導出された割当てパターン候補は，パターン①が処理時間最適，パターン②が課金額最適となっていた．実測結果においても，パターン①はクラウドの計算リソースを利用することでパターン②よりも処理時間が短い結果となり，推測結果と同等の傾向があることを確認できた．一方で，個々の処理時間の推測精度は改善の余地があり，特に性能データの変換方法について以下に示す改良点があると考える．

- 高負荷時における CPU 性能劣化の考慮

- 通信時間推測モデルの改良

**RQ3(効率性)：提案手法の推測は，実機を用いた実測よりも効率良く候補を導出できるか**

提案手法による推測に，2 日程度の時間を要した．おおよその内訳は，ワークフローモデルのプログラム実装に 1 日，割当てパターンの生成と性能値の推測に半日，パレートフロントの導出に半日である．

一方，実機を用いた実測は，エッジの環境構築とクラウドサービスの構築，エッジとクラウド間の通信プログラムの作成，処理時間のログ出力など多くの作業が必要となり，約 7 日程度の時間を要した．

実機を用いて検証する場合は，提案手法による推測とは異なり，実装上の課題を解決しなければならない．例えば，AWS Lambda はデプロイパッケージのサイズに上限があり，サイズの大きい機械学習ライブラリを使用する場合は，当該制限を回避するためコンテナを作成する必要がある．API Gateway と Lambda の接続等，クラウドサービス間の連携方式についても実装する必要があり，接続のための設定や動作確認にも時間を要する．さらに，実システムを用いた検証には，エッジデバイスの調達や，クラウドアカウントやロールの設定など，環境構築に着手する前段階での準備が必要となる場合がある．本実験においては不要であったが，状況によっては上記準備の対応のため，検証期間がさらに長期化する可能性がある．

また，実機を用いた場合，一つの環境で検証できる割当てパターンは基本的に一つのみである．複数の割当てパターンを検証するには，複数のプログラムを作成するか，割当てに応じて処理を柔軟に変更できるプログラム構造にする必要があるため，さらに対応工数が増加する．この観点においても，提案手法を用いた推測の方が，効率的にかつ網羅的に最適な割当てパターンを検証できると考える．

## 6. おわりに

本論文では，物体検知の推論機能を対象としたワークフローモデルを構築することにより，クラウドとエッジで構成されるシステムにおける処理割当て方法を決定する手法を示した．提案手法により，割当てパターンごとに処理時間やリソース利用量を推測した結果をプロットしてパレートフロントを導出することで，最適な割当てパターンの候補を効率良く提示できることを確認した．

今後の展望を以下に示す．

- 他の AI 推論フローに対する適用を通じた汎用性の評価
- クラウドとエッジでの処理時間と通信時間の推測精度向上
- デバイス変更時の影響分析などユースケースに応じた提案手法の応用可能性の検討

## 謝辞

本研究を進めるにあたり，有意義な議論の場を提供頂いた一般財団法人日本科学技術連盟に深く御礼申し上げます．また，議論の場において多くの有益な意見を頂戴した研究会の皆様に感謝いたします．

## 参考文献

[1] IoT Analytics, State of IoT 2022: Number of connected IoT devices growing 18% to 14.4 billion globally, https://iot-analytics.com/number-connected-iot-devices/ (2022/12/27 参照)

[2] Z. Zhou et al., Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing, Proceedings of the IEEE, Vol.107, No.8, P.1738-1762, 2019.

[3] V.Haghighi and N. S. Moayedian, An Offloading Strategy in Mobile Cloud Computing Considering Energy and Delay Constraints, IEEE Access, Vol.6, P.11849-11861, 2018.

[4] W. Zhang and Y. Wen, Energy-Efficient Task Execution for Application as a General Topology in Mobile Cloud Computing, IEEE Transactions on Cloud Computing, Vol.6, No.3, P.708-719, 2018.

[5] K. R. Alasmari et al., Mobile Edge Offloading Using Markov Decision Processes, EDGE 2018, P.80-90, 2018.

[6] H. Wu et al., Collaborate Edge and Cloud Computing With Distributed Deep Learning for Smart City Internet of Thing, IEEE Internet of Things Journal, Vol.7, No.9, P.8099-8110, 2020.

[7] PassMark Software - CPU Benchmark Charts, https://www.cpubenchmark.net/, (2022/12/27 参照)

[8] TensorFlow Hub: Faster R-CNN Resnet-50 V1 640x640, https://tfhub.dev/tensorflow/faster_rcnn/resnet50_v1_640x640/1 (2022/12/27 参照)