ワークフローモデルの構築による AI 推論フローの処理割当て手法の提案

The Offloading Method for AI Inference Flow using a Workflow Model

三菱電機株式会社 設計システム技術センター
Design Systems Engineering Center, Mitsubishi Electric Corporation
○伊藤 弘毅 1)
○Hiroki Itoh

Abstract The allocation of tasks to realize AI inference is a significant problem when the inference is implemented in an IoT system. If the allocation is not proper for the target system, it would cause the problem such as insufficient processing time and expensive cloud billing, which would urge the system to be redesigned for satisfying desired requirements. In this paper, the method which could derive the performance metrics and candidates of the appropriate task allocation. This approach would decrease the burden of deducing the performance and increase the explainability of the task allocation result.

1. はじめに

AI の推論は，前処理，推論処理，後処理の 3 つの処理を順に実行する推論フローとして実現される．クラウドやエッジデバイスなど複数の装置で推論フローを実現する場合，どの処理をどの装置に配置するか割当てを決定する必要がある．その際，各処理の実行時間や処理間のデータ通信量，クラウドサービス利用による課金額などを考慮し，処理の割当てを決定する．十分な検討なしに決定した場合，前述の性能効率性に関する要求品質を満足せず，システム設計とソフトウェア開発をやり直す手戻りが発生する可能性がある．

実機を用いて適切な処理割当てかどうか検証するには，実機上で動作するソフトウェアを開発し，クラウドとエッジデバイスが接続した開発環境を構築しなければならないため，結果を取得するまでに多くの工数を要してしまう．本課題に対処するには，多数考えられる処理割当て候補の中から，処理速度や通信量等の観点から最適な割当て方法を効率良く導出する手法を考案する必要がある．ただし，単純に最適解のみを導出するだけでなく，設計根拠として選択した候補の優位性をステークホルダに共有するため，結果の説明性を持った手法とする必要がある．

本論文では，AI の推論フローをシステム上に実現した時の処理時間やリソース消費量を，実機なしで推測することにより，最適な処理割当て方法を説明性を確保しながら効率的に導出する手法を提案する．上記を実現するため，推論フローの処理と処理間におけるデータの受け渡しを抽象的に表現するワークフローモデルを導入する．提案手法の有効性を確認するため，3 つの推論フローを対象に最適な処理割当てを導出できるか実験した．最後に，実験結果から提案手法の適用による効果と今後解決が必要な課題について述べる．

三菱電機株式会社　設計システム技術センター
Design Systems Engineering Center, Mitsubishi Electric Corporation

兵庫県尼崎市塚口本町 8-1-1　Tel: 06-6497-7355　e-mail:Ito.Hiroki@dr.MitsubishiElectric.co.jp
8-1-1, Tsukaguchi-Honmachi, Amagasaki, Hyogo, Japan
1)三菱電機株式会社　設計システム技術センター　ソフトウェア技術推進部 組込システム開発技術第一グループ
Design Systems Engineering Center, Mitsubishi Electric Corporation

【キーワード：】AI, 推論, IoT, ワークフローモデル, 処理割当て

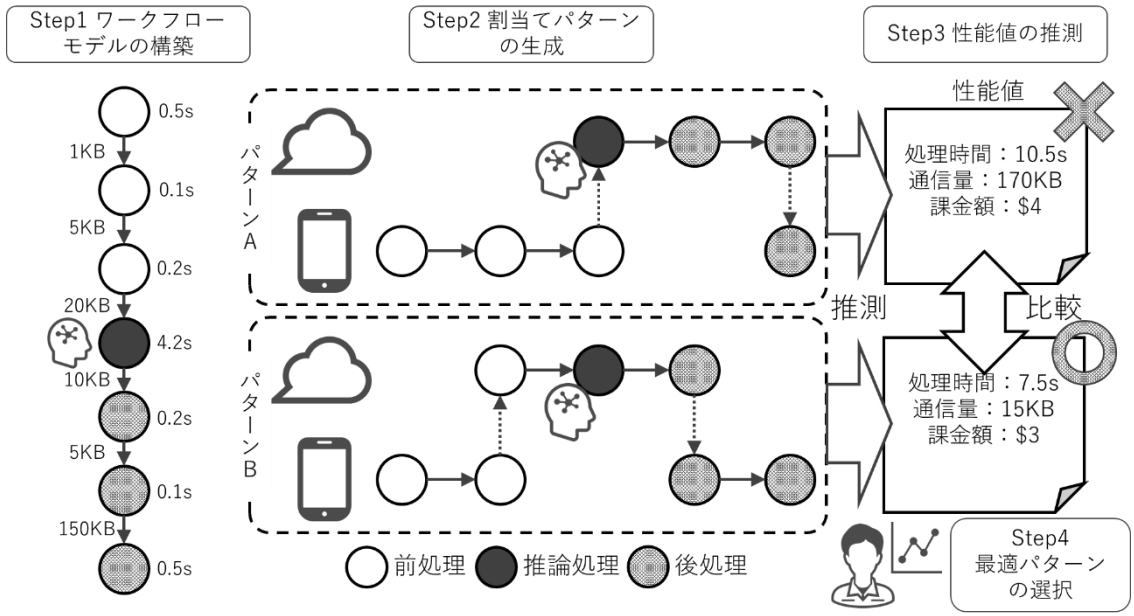


図 1　提案手法の全体像

2.　課題

IoT システムは，利用できる計算リソースが異なる複数種類の装置が連携して処理を実行することにより，ユーザに価値ある機能を提供する．IoT システムを構成する装置の種類は様々だが，処理時間等の性能効率性とコストのトレードオフを考慮し，装置のスペックを決定していく必要がある．すなわち，クラウドで処理を実行する場合は潤沢な計算リソースを利用できる一方，使用に伴うランニングコストが発生する．一方，エッジデバイスで処理を実行する場合は，多くの計算リソースは見込めないが，クラウドに送信する通信のオーバーヘッドが抑えられるのに加え，クラウドの利用コストは発生しない．

AI の推論処理は，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追加することで実現される．AI の推論機能はこれらの処理を順番に実行することで，必要とされる結果を出力する．

IoT システムに推論フローをデプロイする時には，前述のトレードオフを考慮して，推論フローを構成する各処理を装置に適切に割当てる必要がある．考慮を怠った場合，システムテスト時または運用時に意図せず装置間で大量の通信が発生したり，高額なクラウド課金が発生したりするなどの問題が起こる可能性がある．この問題が発生すると，推論フローの実装を見直す必要が出てくるため，ソフトウェアを再設計する手戻りが発生してしまう．

上記課題に対して，数理アルゴリズムを適用することにより，システムに推論フローを実装する前に処理の割当てを机上演算で導出する既存研究が存在する[1][2]．しかし，これらの方法では，最終的な最適化結果を知ることができるが，その結果が他の割当て方法と比較して，どの程度優位なものであるかを確認することができない．実開発では，ステークホルダへの設計根拠の説明のため，複数の割当て候補を提示して，対象のシステムに対してどの候補が適切かを示す必要が出る可能性がある．また，今回対象とするのは処理時間やクラウド課金額などトレードオフの関係にある複数の要素を最適化する多目的最適化問題である．この場合，システムに求められる処理時間やクラウド課金額の要求割合をパラメータで表現する必要が出てくるが，開発者が正確にパラメータの値を設定することが難しい可能性もある．

3.　提案手法

本手法では，AI の推論フローを表現するワークフローモデルを構築することで，フローを構成する処理を各装置に最適に割当てる方法を導出する．その際，ステークホルダへの説明性を確保するため，最適解と他候補の結果の差異を提示することを特徴とする．

本手法の全体像を図 1 に示した。最適な処理割当ては,以下の 4 つのステップから導出される．

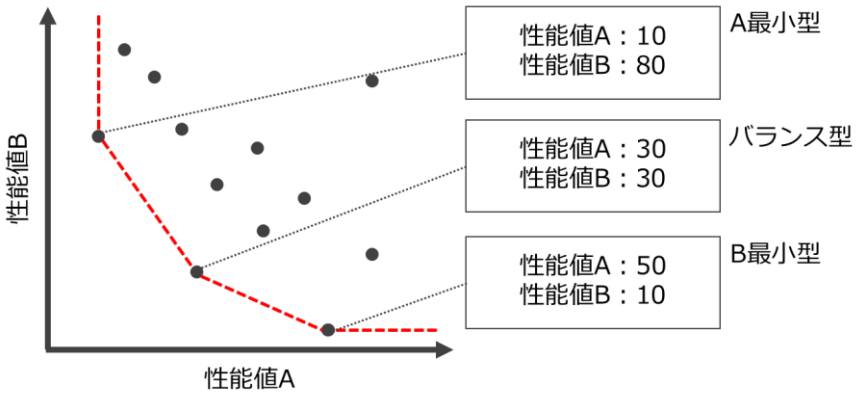


図 2　パレートフロントと最適解候補の導出例

Step1：ワークフローモデルの構築

ワークフローモデルは，図　1 の Step1 に示したような AI の推論フローで実行される処理をノード，処理の間で渡されるデータの流れをエッジとして表現したモデルである．ノードに汎用 PC 上で動作させた時に実測した処理時間を，エッジに処理間で渡されるデータ量を設定する．

Step2：割当てパターンの生成

ワークフローモデルを構成する各処理をクラウドまたはエッジデバイスに割当てたパターンを生成する．はじめに，推論フローの中で実行装置を固定する処理を定義する．その後，固定しない各処理を両者に割当てる際に考えられる全ての組合せを，割当てパターンとして抽出する．

Step3：性能値の推測

生成した各割当てパターンについて，システム上に推論フローを実装した際の性能値を推測する．本手法で推測する性能値は，処理時間と通信時間，クラウド課金額である．

処理時間は，ノードに設定された実測処理時間に対し，実測マシンとクラウドまたはエッジデバイスのマシン性能比で補正する．計算式を以下に示す．補正に使用するマシン性能値として，搭載 CPU のクロック周波数や PassMark[3]等のベンチマークの値，コア数を利用する．

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

通信量は，処理の間で実行装置が切り替わるときのデータ量を積算して算出する．通信量は，クラウド課金額を導出するため，通信方向を区別して算出する．

通信時間は，通信量と想定される通信速度からデータ転送に要する時間を算出する．計算式は以下の通り．

$$通信時間 = \frac{通信データサイズ}{通信速度}$$

クラウド課金額は，利用するクラウドサービスの料金体系を参照し，推測した処理時間や通信量を代入して導出する．

Step4：最適パターンの選択

推測した性能値をグラフにプロットし，パレートフロントを導出することで，最適な処理割当て候補を抽出する．パレートフロントは，推測された性能値について他のどの解にも優越されない最適解を繋げたトレードオフ曲線である．性能値をプロットして得られるパレートフロントの例と最適解の候補の例を図 2 に示した．設計者は得られた候補の性能値を確認し，その中から要求性能を満足する候補を選択する．

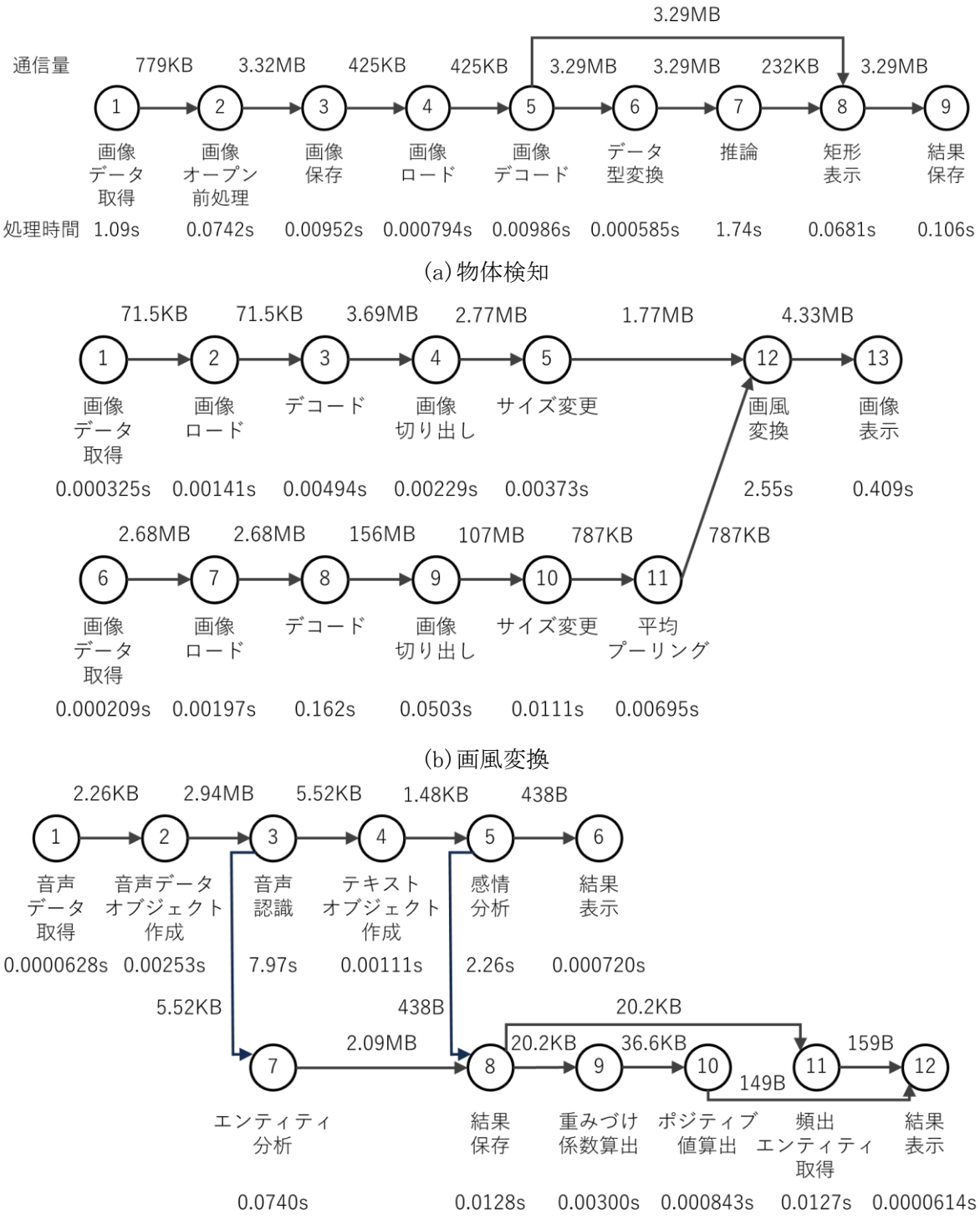


図 3　物体検知のワークフローモデル

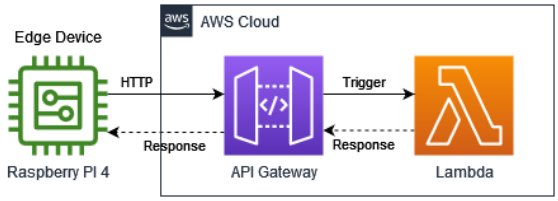
4.　実験

4.1　概要

3 章で記した提案手法が有効かケーススタディで実験した．本実験では，3 つの推論フローを定義し，本手法により処理割当ての最適解を導出することができるか確認した．各々の推論フローの概要を以下に示す．また，推論フローに対応するワークフローモデルを図 3 に示す．当該ワークフローに，実験 PC で測定した処理時間と処理間で発生した通信量を記載した．

(a) 物体検知の推論フロー

入力画像の物体位置と種類を推論[4]し，矩形で囲った画像を出力する推論フローである．



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 4　本実験のシステム構成

(b) 画風変換の推論フロー
　入力画像を指定したスタイル画像の画風に変換[5]する推論フローである.
(c) 音声認識と感情分析の推論フロー
　音声を認識[6]し，テキスト化した発言内容を感情分析[7]した結果を表示する推論フローである.

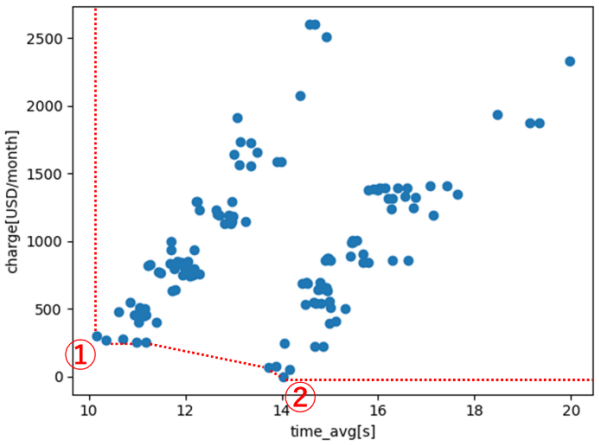
　筆者は，上記 3 つの推論フローを図 4 に示すシステム構成にデプロイすることを想定した時の処理時間と通信量を推測した．また，処理時間とクラウド課金額を評価軸に，最適な処理割当てを導出することを試みた.
　本実験では，実測 CPU の処理時間から実機 CPU の処理時間への変換に，PassMark の整数演算または浮動小数点演算の性能値を採用した．これは，推論フローにおける主要な処理である推論処理が数値演算を実行するためである．具体的には，(a)と(c)の推論フローには Integer Math，(b)の推論フローには Floating Point Math の性能値を使用した．また，マルチスレッドで動作する推論処理に対応するため，処理実行時に使用されるコア数を CPU の性能比の算出要素として追加した．具体的な計算式を以下に示す.

$$推測処理時間 = 実測PC処理時間 \times \frac{実測CPU\ PassMark性能値(1\ コア相当) \times 実測PC実行コア数}{実機CPU\ PassMark性能値(1\ コア相当) \times 実機PC実行コア数}$$

　通信量は，処理の間で実行装置が変わる場合に発生するものとし，推論フロー全体の通信量として加算する．クラウド―エッジデバイス間のネットワーク速度は 100Mbps として推測した.

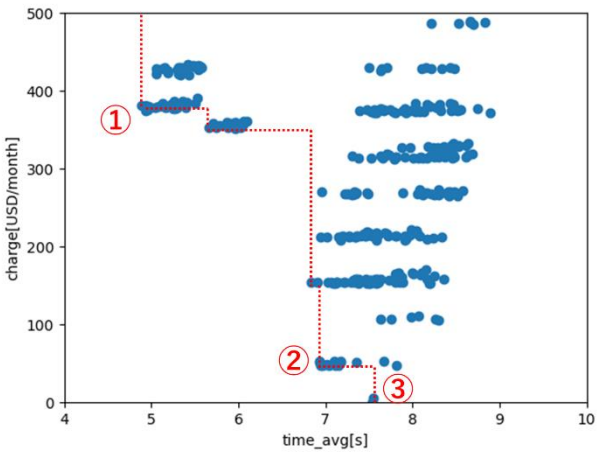
4.2　結果

　各推論フローに対する実験結果を図 5 に示す．推測した性能値をプロットしたグラフに，パレートフロントを赤線で示している.
　(a)では，処理 1 と 9 をエッジデバイスに割当て，それ以外の処理をクラウドかエッジデバイスに割当てた場合の全 128(=2^7)パターンについて性能値を推測した．パレートフロントを確認した結果，割当て可能な処理を全てクラウドに配置して処理時間を最小にするパターンと，全てエッジに配置してクラウド課金額を最小にするパターンの 2 つが最適解の候補として導出できた.
　(b)では，処理 1 と 13 をエッジデバイスに割当て，それ以外の処理をクラウドかエッジデバイスに割当てた場合の全 2048(=2^{11})パターンについて性能値を推測した．パレートフロントを確認したところ，割当て可能な処理を全てクラウドまたはエッジデバイスに割当てるパターンと，その中間解を含めた 3 つの最適解の候補を導出することができた.
　(c)では，処理 1, 6, 12 をエッジデバイスに割当て，それ以外の処理をクラウドかエッジデバイスに割当てた場合の全 512(=2^9)パターンについて性能値を推測した．パレートフロントを確認した結果，割当て可能な処理を全てクラウドまたはエッジデバイスに割当てたパターンに，2 つの中間解を含めた 4 つの最適解の候補を導出することができた.
　上記のとおり，本手法を用いると，複数の最適解の候補について性能値を比較できるため，ステークホルダへの説明性を担保しながら，処理の割当てパターンを決めることができると考える.



	パターン①	パターン②
処理割当て	クラウド：2～8 エッジ：1、9	クラウド：なし エッジ：1～9
処理時間 (平均)	10.15s	14.04s
課金額	312USD/month	0USD/month

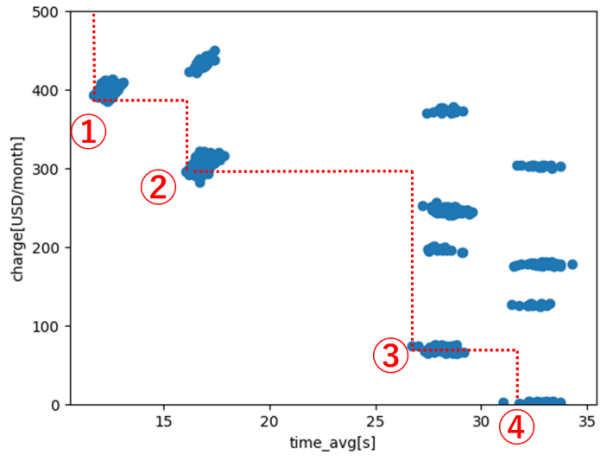
(a) 物体検知



	パターン①	パターン②
処理割当て	クラウド：2～12 エッジ：1,13	クラウド：6～11 エッジ：1~5,12,13
処理時間 (平均)	4.88s	6.95s
課金額	381.0USD/month	47.4USD/month

	パターン③
処理割当て	クラウド：なし エッジ：1～13
処理時間 (平均)	7.53s
課金額	0USD/month

(b) 画風変換 (抜粋)



	パターン①	パターン②
処理割当て	クラウド：2～5,7～11 エッジ：1,6,12	クラウド：2～4,7～11 エッジ：1,5,6,12
処理時間 (平均)	11.7s	16.1s
課金額	393.0USD/month	295.3USD/month

	パターン③	パターン④
処理割当て	クラウド：4~5,7~8 エッジ：1~3,6,9~12	クラウド：なし エッジ：1~12
処理時間 (平均)	27.0s	31.8s
課金額	74.0USD/month	0USD/month

(c) 音声認識と感情分析 (抜粋)

図 5　割当てパターンの処理時間と課金額

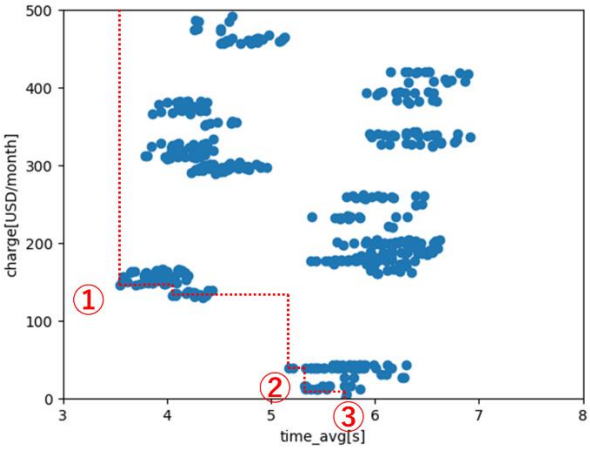
5.　考察

5.1　精度

本手法による性能値の推測結果の精度を確認するため，実機で図 4 に示したシステムを構築し，推論フロー(a)の各処理時間の実測値を取得し，両者を比較した．各処理における推測値と実測値を表 1 に示す．パレートフロントにより最適解の候補として導出されたパターン①とパターン②を対象に比較した．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

表 1　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



	パターン①	パターン②
処理割当て	クラウド：2～12 エッジ：1,13	クラウド：6～11 エッジ：1~5,12,13
処理時間 (平均)	3.58s	5.33s
課金額	149.2USD/month	11.8USD/month

	パターン③
処理割当て	クラウド：なし エッジ：1～13
処理時間 (平均)	5.71s
課金額	0USD/month

図 6　(b)画風変換における画像サイズ変更時の推測結果

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，本手法の通信時間がネットワークの輻輳を考慮しておらず，データ量とデータ帯域から単純に転送所要時間を算出したことに起因すると考えられる．これは，ネットワーク輻輳を表現したモデルで通信時間を推測するようにすることで，通信時間の予測精度が向上する可能性がある．

パターン②の処理 7 は，予測値よりも実測値の方が長い結果となった．処理 7 は推論処理に該当する．この誤差は，マルチスレッドで動作する推論処理によりエッジデバイスの 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．

5.2　効率性とスケーラビリティ

上記の実験で割当てパターンの性能値を取得するまでに，本手法による推測の場合は 2 日，実機を用いた実測の場合は 7 日要したため，本手法の方が効率良く最適な割当て方法を導出できると考える．実機を用いた時に長い時間を要するのは，実装上の課題を解決する必要があるためである．例えば，クラウドにアップロードするアプリケーションのサイズ制限や，連携サービスの設定，アカウントやロールの準備等である．また，実機を用いる場合は，基本的に一度に一つの割当てパターンしか性能値を測定できない．

なお，本実験において，ワークフローモデルの定義後，パレートフロントを導出するまでに要する計算時間は 5 分程度であった．今回の実験で用いたワークフローモデルの規模で，各処理をエッジデバイスまたはクラウドに割当てる問題であれば，全ての割当てパターンについて性能値を推測し，パレートフロントで最適なパターン候補を導出できると考える．

一方，本実験ではクラウドとエッジデバイスで構成されるシステムについて，推論フローの各処理を割当てる前提で実験をした．想定システムを構成する装置の種類が増えた場合，割当てパターンの総数は一気に増大する．例えば，最もパターンの組合せ数が多くなる(b)の推論フローにおいて，クラウドとエッジデバイスの間にゲートウェイ 1 台を接続する構成の場合，割当てパターンの総数は 177147 ($=3^{11}$) となる．上記条件でパレートフロントの導出を試みた場合，実験 PC で結果が得られるまでに約 5 時間を要した．本手法は，割当て対象となる装置数が少ない時は，全組合せの割当てパターンの性能値を推測することができるが，装置数が増えたり推論モデルを切替えたりすることで組合せ数が増大する場合におけるスケーラビリティに課題があると考える．

本課題については，全ての割当てパターンを推測するのではなく，最適解を導出する可能性の高い割当てパターンの性能値を優先的に算出できるようにアルゴリズムを改良する対応策が考えられる．

5.3 仕様変更時の影響分析

本実験の(b)画像変換の推論フローについて，入力画像サイズを小さくした時に推測結果に与える影響を確認した．画像サイズ変更時の推論フローの最適化結果を図 6 に示す．図 5(b)と比較すると，性能値のプロットの傾向は変わらず，導出された最適解の候補も同じであることが分かる．しかし，画像サイズを小さくしたことにより，全体的に処理時間が短縮され，課金の推測額も小さくなっていることから，与えられた要求を考慮して最終的に採用する候補が変化する可能性がある．

上記のように，ワークフローモデルに設定するパラメータを変更することにより，仕様変更時における処理割当てへの影響を検証することができる．例えば，大きな画像サイズを入力したときの処理時間の推測結果が想定よりも長かった場合，小さな画像サイズに対応するパラメータで再検証することにより，入力画像の解像度と処理時間のトレードオフを考慮しながら最適な設計を検討できるようになると考える．

6. おわりに

本手法により，複数装置から構成されるシステム上に AI の推論フローを実現する時の処理時間やクラウド課金額等を机上で推測し，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の候補を導出できることを確認した．

今後の展望を以下に示す．

- 実機における処理時間と通信時間の推測精度向上
- スケーラビリティを考慮した効率的な最適化アルゴリズムの検討
- ユースケースに応じた提案手法の応用可能性の追加検討

謝辞

本論文の内容は 2022 年度 SQiP 研究会の研究コース 5 での活動で議論して洗練させていきました．執筆にあたり，石川冬樹主査，栗田太郎副主査，徳本晋副主査には丁寧に指導を賜りました．深く御礼を申し上げます．

参考文献

[1] 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.
[2] 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.
[3] PassMark Software - CPU Benchmark Charts, https://www.cpubenchmark.net/, (2022/12/27 参照)
[4] TensorFlow Hub: Faster R-CNN Resnet-50 V1 640x640, https://tfhub.dev/tensorflow/faster_rcnn/resnet50_v1_640x640/1 (2022/07/31 参照)
[5] TensorFlow Hub: arbitrary-image-stylization-v1-256, https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/2 (2023/07/31 参照)
[6] SpaCy: en_core_web_sm, https://spacy.io/models/en (2023/07/31 参照)
[7] TensorFlow Hub: bert_en_uncased_L-12_H-768_A-12, https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/3 (2023/07/31 参照)