Vol.2026-ARC-264 No.38
Vol.2026-SLDM-211 No.38
Vol.2026-EMB-71 No.38
2026/3/16

Jetson Orin Nanoにおける不確実性駆動型予見予測モデルの評価とFP16精度計算の検討

朱 允楷[1,a)]　昼間 彪吾[1]　顧 茗瑞[1]　伊藤 洋[1]　尾形 哲也[1]　北村 俊明[1]　木村 啓二[1]

概要：不確実性が高い環境下でロボットが安全かつ頑健に行動するために，不確実性駆動型先見予測（Uncertainty-Driven Foresight Prediction）に基づく動作生成モデルが提案されている．一方で，先見予測では 1 制御周期内に複数回の推論を要するため計算負荷が高く，計算資源と電力が制約されたエッジデバイス上でリアルタイムに動作させる上での課題となる．本稿では，画像と関節角度を入力として関節角度を予測する CAE（Convolutional Autoencoder）と Stacked RNN から構成される UF-RNN モデルを，NVIDIA Jetson Orin Nano へ移植し，ドア開扉タスクにおけるリアルタイム動作の実現可能性を検証した．推論ライブラリを TensorRT へ変換することで推論遅延のばらつきを抑制し，10Hz の制御周期で安定して動作することを確認した．また，デスクトップ PC 環境と比較した電力計測により，低消費電力のエッジ環境において同等のタスク実行が可能であることを示した．さらに，深層学習で広く利用されている FP16 精度計算の適用が電力効率・演算効率・成功率に与える影響を評価した．その結果，FP16 により消費電力は低減する一方で，動的レンジを大きく必要とする重み分布や小規模行列演算を含む構成では速度向上が限定的，または低下し得ること，また量子化誤差が成功率に影響し得ることを確認した．さらに，シミュレーション環境において成功率評価を行い，TensorRT への変換および混合精度が成功率に与える影響を定量化した．以上より，先見予測型動作生成モデルのエッジ実装における推論基盤設計と精度選択に関する指針を示す．

Performance and FP16 Precision Evaluation of Uncertainty-Driven Foresight Prediction Model on Jetson Orin Nano

1. はじめに

国際的に少子高齢化が進行しており，これに伴い労働力不足が深刻化している．そのため，介護・物流・製造など様々な分野でロボットの導入が進んでいる．これらのロボットは，カメラやセンサーから得られる環境情報に基づき，動作生成モデルを用いて適切な行動を選択・実行することが求められている．このような環境に応じた自律行動を実現する動作生成手法として，深層予測学習モデルを用いたデモンストレーション学習（Learning from Demonstration; LfD）が注目されている [1]．深層予測学習は，人間による教示動作データを用いて，深層学習モデルを学習し，固定の学習済みモデルを用いて将来の動作を予測・生成する手法である．

しかしながら，現実世界は理想的な環境とは異なり，不確実性や変動要素が多く存在する．例えば，ドアの開閉動作を学習したロボットが，異なる形状のドアに遭遇した場合，学習データに基づく予測が困難となり，動作生成の失敗につながる可能性がある．このような課題に対し，不確実性駆動型先見予測（Uncertainty-Driven Foresight Prediction）に基づく UF-RNN モデルが提案されている [2], [3].

UF-RNN に基づく制御では，Convolutional Neural Network（CNN）を用いてカメラ画像から特徴量ベクトルを抽出し，さらに Recurrent Neural Network（RNN）を用いてロボット関節角度と画像特徴量から将来の動作を予測する．1 制御周期内では，多数回の RNN 推論を繰り返し実行し，将来の状態を先読みすることで不確実性の小さい将来へ誘導する．深層学習推論時の計算複雑性が高く，かつ小行列の演算が多いため，計算資源や電力の制約が厳しいエッジ環境へ展開する上での課題となる．

[1] 早稲田大学
Waseda University
[a)] yunyun3099@moegi.waseda.jp

本稿では，UF-RNN モデルをエッジコンピューティングの観点から低消費電力で実行することを目的とする．具体的には，UF-RNN モデルを NVIDIA Jetson Orin Nano[4] へ移植し，不確実性を考慮した複雑な推論と予測処理を低消費電力かつ小型なエッジデバイス上でリアルタイム実行できることを検証する．さらに，深層学習処理で広く用いられている FP16 精度計算の適用が，上記 end-to-end ロボット制御系の電力効率・演算効率・成功率に与える影響を評価し，先見予測型動作生成モデルのエッジ実装における精度選択の指針を得る．

本研究の主な貢献を以下にまとめる．

- UF-RNN モデルを Jetson Orin Nano へ移植し，実機制御周期 10Hz でのリアルタイム動作可能性を示した．
- TensorRT[5] による推論最適化により，推論遅延のばらつきを抑制し，リアルタイム実行を実現した．
- 電力計測および推論速度計測により，デスクトップ PC 環境との比較を通じて Jetson 上での実行の有効性を定量化した．
- FP16 適用時の利点（消費電力低減）と限界（構成によって速度向上が限定的／量子化誤差の影響）を示し，その要因を考察した．
- シミュレーション環境において成功率評価を行い，TensorRT への変換および混合精度が成功率に与える影響を定量化した．

第 2 節では UF-RNN モデルの概要と構成要素を述べる．第 3 節では Jetson Orin Nano への実装方針と計測方法を述べる．第 4 節では実機環境での推論遅延・消費電力の評価結果と，FP16 適用に関する考察を示す．第 5 節ではシミュレーション環境での成功率評価を行い，精度選択が成功率に与える影響を述べる．

2. UF-RNN モデルによるロボット動作生成

2.1 UF-RNN モデルの概要

Learning from Demonstration (LfD) は，ロボットが強化学習により報酬を最大化するのではなく，人間によるデモンストレーションから動作生成モデルを学習する枠組みであり，ロボット制御で広く用いられている [6]．LfD の動作生成モデルとして，現在のカメラ画像や関節角度を入力として，将来のカメラ画像や関節角度を予測する深層学習モデルが提案されている [1], [7]．しかし，現実世界は理想的な環境とは異なり，不確実性や変動要素が多く存在する．例えば，ドアの開閉動作を学習したロボットが，異なる形状のドアに遭遇した場合，学習データに基づく予測が困難となり，動作生成の失敗につながる可能性がある．この課題に対し，不確実性駆動型先見予測（Uncertainty-Driven Foresight Prediction）に基づく UF-RNN モデルが提案されている [2], [3].

UF-RNN モデルは，カメラ画像を低次元の特徴量に変換する CAE（Convolutional Autoencoder）と，時系列予測を行う Stacked RNN から構成される．推論処理は，カメラ画像と関節角度を入力として次時刻の関節角度を出力する．図 1 に推論パイプラインの構成を示す．CAE は推論時にはエンコーダ部のみを使用し，64×64 ピクセルの RGB 画像を 32 次元の特徴ベクトルへ変換する．Stacked RNN は，Fast Image RNN，Fast Joint RNN，Slow RNN の 3 種類の役割の異なる LSTM（Long Short-Term Memory）[8] モジュールを階層的に組み合わせた構造を持つ．

まず入力画像は CAE エンコーダにより低次元特徴量へ変換され，現在の関節角度と併せて Stacked RNN へ入力される．Stacked RNN 内部では，不確実性を考慮するために隠れ状態へノイズを加えた 5 種類の系列を生成し，Foresight モジュールにより 5 ステップ先までの内部予測を行う．得られた予測系列の分散の減少量に基づいてノイズ系列を選択し，選択された系列に対応する内部状態を用いて次時刻の関節角度予測を出力する．この推論フローに対応するよう，以下ではまず CAE の構成を述べ，続いて Stacked RNN の構成と不確実性処理を説明する．

2.2 CAE の構成

CAE は，畳み込みニューラルネットワーク（Convolutional Neural Network; CNN）を用いて画像を低次元の特徴量ベクトルに変換する．エンコーダは複数の畳み込み層と全結合層から構成される．デコーダはエンコーダの逆構成であり，特徴量ベクトルから元の画像を再構成する．推論時にはデコーダを用いず，エンコーダのみを使用する．

入力には，サイズ 64×64 の RGB 画像が用いられる．最初に 2 次元畳み込み層が 4 層連続し，その後，全結合層が 3 層と連続する．アクティベーション関数には，Leaky ReLU 関数を用いている．Batch Normalization 層が全結合層の途中に挿入されている．最終的に，32 次元の特徴量ベクトルが出力される．この特徴量が Stacked RNN への入力となる．

2.3 Stacked RNN の構成

Stacked RNN は，Foresight モジュール，内部状態にノイズを与える処理，および次の時刻の関節角度を予測する推論モジュールから構成される．Foresight モジュールでは，RNN モジュールの隠れ状態に 5 種類のガウスノイズを加えたのち，それぞれのノイズを加えた隠れ状態を用いて 5 回の推論ループを実行することによって，時刻 $t+1$ から $t+5$ の状態を予測する．予測系列から所定のステップの予測を取り出し，`image_var` と `joint_var` を元状態と比較して分散の減少量を算出する．分散減少が最大となるノイズを選択し，そのノイズを用いた隠れ状態を次の時刻の RNN モジュールの入力として使用する．この Foresight モジュールにより，不確実性を考慮した動作生成が可能と

Vol.2026-ARC-264 No.38
Vol.2026-SLDM-211 No.38
Vol.2026-EMB-71 No.38
2026/3/16

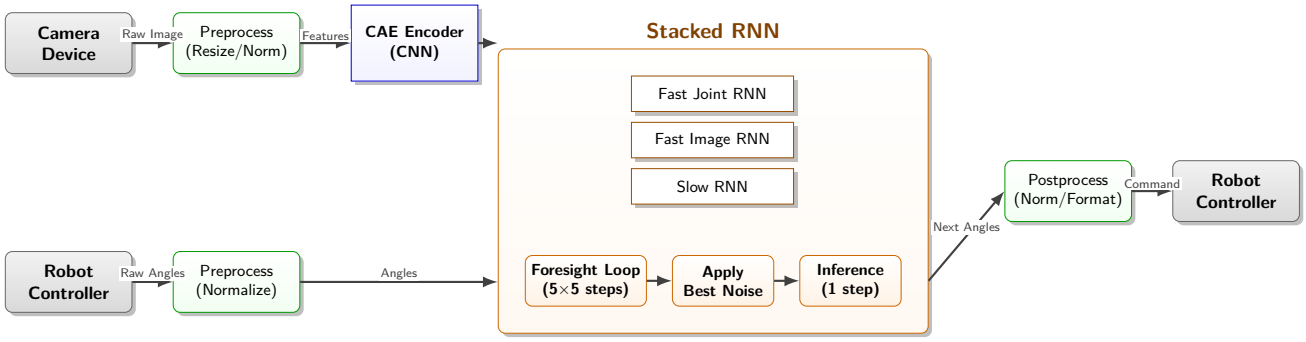


図 1: UF-RNN の推論パイプライン

なる．

Stacked RNN の RNN モジュールは，Long Short-Term Memory (LSTM)[8] を用いている．LSTM セルは，入力ゲート，忘却ゲート，出力ゲートの 3 つのゲートと隠れ状態から構成される．各ゲートはシグモイド関数を用いて計算され，隠れ状態は tanh 関数を用いて更新される．

RNN モジュールでは，Fast Image RNN，Fast Joint RNN, Slow RNN の 3 種類が用いられる．各 RNN モジュールのモデル構造をそれぞれ図 2a–2c に示す．モデルの入力を赤字，出力を青字で示す．入出力で同名の行列を用いている場合，語尾に_in，_out を付けた．モデル入出力，各層入出力の行列サイズを batch×size という形式で示す．アクティベーション層などの入出力で行列サイズが不変の層については，上の層から行列サイズが伝搬される．Fast Image RNN は，CAE のエンコーダから得られた画像特徴量と Slow RNN の隠れ状態を入力とし，次の時刻の画像特徴量と Fast Image RNN の隠れ状態を予測する．Fast Joint RNN は，現在の関節角度と Slow RNN の隠れ状態を入力とし，次の時刻の関節角度と Fast Joint RNN の隠れ状態を予測する．Slow RNN は，Fast Image RNN と Fast Joint RNN の隠れ状態を入力とし，次の時刻の Slow RNN の隠れ状態を予測する．各 RNN モジュールの出力隠れ状態は，次の時刻の各 RNN モジュールの入力として使用される．これらの RNN モジュールを組み合わせることで，時刻 t の画像特徴量と関節角度から時刻 $t+1$ の関節角度を予測する．

5 種類のノイズ系列を同時に評価するため Foresight 時の batch size は 5 であり，次時刻の関節角度を予測する推論モジュールでは batch size が 1 である．各モジュール内部の行列サイズは小規模であり，入力行列サイズは最大でも batch size が 5 のときの Slow RNN の隠れ状態（図 2a, 2b の slow_h，図 2c の slow_h_in，slow_c_in）で 5×60 に留まる．したがって，Stacked RNN 全体として小行列演算が 1 推論周期内に多数回発生する点が計算特性の重要な特徴である．

3. Jetson Orin Nano への実装

3.1 評価対象

本稿では，デスクトップ PC 環境で動作していた UF-RNN モデルによるロボット制御プログラムを Jetson Orin Nano へ移植し，実機制御（10Hz）で動作可能かを検証した．文献 [2], [3] のデスクトップ PC 上のドア開け実験で使用された UF-RNN プログラム一式を用いた．デスクトップ PC と Jetson Orin Nano の諸元を表 1 に示す．

3.2 TensorRT による推論最適化と精度設定

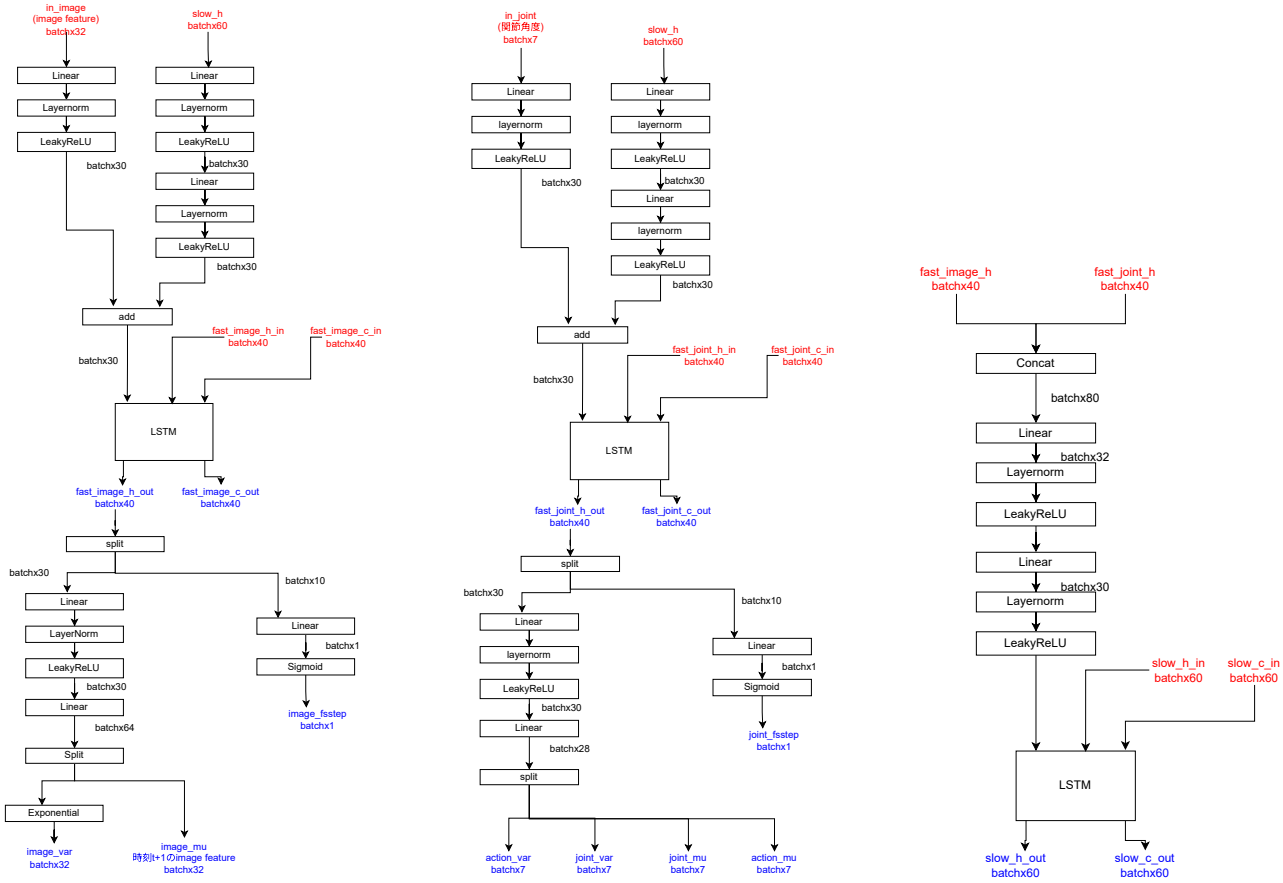
本研究では TensorRT[5] を用いて推論処理を最適化し，特にエッジ環境における遅延のばらつき低減とリアルタイム実行を狙った．TensorRT は，NVIDIA 製の高性能深層学習推論ライブラリであり，モデルの量子化，モデル内のレイヤー融合，および GPU アーキテクチャに最適化されたカーネルの使用を通じて，推論速度の向上を実現する．

また，本稿で比較のベースラインとする UF-RNN のオリジナル実装は，文献 [2], [3] で用いられた PyTorch 実装であり，推論は FP32 で実行される．このベースラインに対して，TensorRT へ変換した際の推論遅延・電力・成功率の変化を評価するため，以下のバックエンド／精度設定を比較した．

- PyTorch FP32（ベースライン実装）
- TensorRT FP32
- TensorRT FP16

なお，INT8 等の整数量子化については本稿では評価対象外とした．精度面では，ロボット制御が時系列タスクであるため，ある時刻の入力や出力の微小なずれが以降の状態遷移を変化させ，誤差が累積・増幅して最終的な成功率へ影響し得る．このため，量子化誤差のみを要因として切り分けて評価することが難しい．また，量子化誤差が出力関節角度の誤差やタスク成功率へ直接影響し得ることに加え，TensorRT の INT8 では代表データによるキャリブレーションが必要であり，入力分布・環境条件に依存し

Vol.2026-ARC-264 No.38
Vol.2026-SLDM-211 No.38
Vol.2026-EMB-71 No.38
2026/3/16



(a) Fast Image RNN　　(b) Fast Joint RNN　　(c) Slow RNN

図 2: Stacked RNN を構成する各 RNN モジュールの構成

表 1: 使用機材の諸元

項目	Desktop PC	Jetson Orin Nano
CPU	12th Gen Intel Core i7-12700F (12 コア/20 スレッド)	Cortex-A78AE (6 コア)
CPU クロック	最大 4.9 GHz	1.73 GHz
GPU	NVIDIA GeForce RTX 3060 Ti (8 GB)	NVIDIA Ampere architecture GPU [Integrated]
GPU 世代	Ampere 世代	Ampere 世代
GPU クロック	Boost 1.67 GHz	1.02 GHz
CUDA コア数	4864	1024
Tensor コア数	152	32
メモリ	32 GB DDR4	8 GB LPDDR5

てスケールが変動すると精度劣化のリスクが高い．また，Stacked RNN は Foresight により小規模な行列演算を多数回反復して実行する構造であり，INT8 化しても量子化／逆量子化や要素演算のオーバヘッドが支配的となって速度向上が限定的となる可能性がある．以上より，本稿ではまず FP16/FP32 に絞って，遅延・電力・成功率への影響を体系的に評価した．

TensorRT FP16 実装において，実機評価では，推論パイプラインを FP16 に統一し，前処理段階で画像・関節角度の入力を FP16 へ型変換してから CAE+RNN へ入力する構成とした．シミュレーション評価では，CNN（CAE）部は FP32 のまま，RNN 部のみを FP16 で動作させる混合精度構成（RNN FP16）で評価した．混合精度構成（RNN FP16）では，シミュレータのカメラ画像入力を FP32 のまま CAE encoder へ入力し，CAE encoder の出力特徴量に対して RNN 入力前に FP16 への型変換を行った．一方で関節角度系列は，前処理（preprocess）段階で FP16 へ型変換した．これは，CAE encoder の全結合層において活性値

Vol.2026-ARC-264 No.38
Vol.2026-SLDM-211 No.38
Vol.2026-EMB-71 No.38
2026/3/16

(a) 開扉前（初期位置）　(b) 開扉後（pull type）

図 3: 実機におけるドア開け動作（pull type）の様子

のダイナミックレンジが広く，FP16 ではオーバーフローが発生する可能性があったためである．シミュレータでは制御周期の推論周期の時間的制約がなく，CAE encoder を FP32 のまま動作させても評価が可能であり，数値安定性を優先して上記の混合精度構成を採用した．

4. 実機評価

4 節では，UF-RNN モデルを用いたロボット制御プログラムを Jetson Orin Nano 上で実行し，制御周期 10 Hz に対する推論遅延のリアルタイム性と安定性，および推論実行時の消費電力を評価する．計測では，図 3 のように，実際にロボットアームを用いてドア開け動作を行いながら推論遅延と電力を計測した．

図 3 に，実機でのドア開け動作の様子を示す．開扉前には，ロボットアームがドアハンドル前方まで接近し，グリッパでハンドルを把持する．開扉後には，把持状態を維持したままハンドルを手前方向へ引くことでドアを回転させ，ドアが開いた状態へ遷移する．

4.1 計測方法

4.1.1 推論遅延

UF-RNN モデルの推論は，CAE エンコーダによる特徴量抽出と，Stacked RNN による時系列推論からなる 2 段構成（CAE+RNN）である．各制御周期 t において，観測取得（画像・関節角度）後に実行される推論処理を対象とし，CAE エンコーダ処理時間 T_{CAE} と Stacked RNN 処理時間 T_{RNN} を計測した．ここで T_{RNN} には，Foresight モジュールに伴う複数回の RNN 推論（5 サンプル ×5 ステップ）と，次時刻関節角度予測を含める．

ハードウェア立ち上げの時間を無視するため，推論遅延は，連続実行した $N = 200$ フレームのうち初回フレームを除外し，残り $N - 1 = 199$ フレームを評価対象とした．評価対象フレーム集合 $\{2, \ldots, N\}$ に対し，平均遅延と最大遅延を以下で定義する．

$$\overline{T} = \frac{1}{N-1} \sum_{i=2}^{N} T_i, \quad T_{\max} = \max_{i \in \{2,\ldots,N\}} T_i.$$

本稿では T_{CAE} および T_{RNN} についてそれぞれ $\overline{T}$ と $T_{\max}$ を報告する．

4.1.2 消費電力

デスクトップ PC 環境および Jetson 環境で電力計測を行い，推論バックエンド（PyTorch/TensorRT）による消費電力の違いを比較した．デスクトップ PC 環境では Intel Running Average Power Limit (RAPL) を用いて CPU の消費電力を計測し，NVIDIA Management Library (NVML) を用いて GPU の消費電力を計測した．Jetson 環境では Jetson の組み込み電力センサーを用いてシステム全体の消費電力を計測した．各環境とも，推論（CAE+RNN）を連続実行している区間における平均消費電力を算出した．また比較のためアイドル時の消費電力も計測した．

4.2 計測結果

4.2.1 推論遅延

表 2 に，推論（CAE+RNN）を構成する各モジュールの推論時間（平均・最大）を示す．

制御周期 10 Hz は 100 ms である．Jetson 上で PyTorch 推論（CAE+RNN）を行う場合，特に Stacked RNN 部の遅延が大きく，制御周期を超過することが分かる．実際に Jetson PyTorch GPU では RNN の最大遅延が 132.153 ms であり，100 ms を超過するため制御ループのオーバーランが発生する．一方，TensorRT では平均遅延・最大遅延が大きく改善し，RNN 部の最大遅延は約 23 ms（FP32）まで低下するため，前処理・後処理や制御計算を含めても十分なマージンを確保できる．また，TensorRT により平均値だけでなく最大値も改善していることは，演算融合やカーネル最適化により GPU 起動オーバヘッドおよび実行のばらつきが抑制されたことを示している．

一般に，FP16 など低い演算精度はメモリ転送量の削減や Tensor Core による並列演算の利用により高速化しやすい．表 2 においても CAE エンコーダ部は FP16 が FP32 よりわずかに高速であり（CAE avg: 1.075 ms → 1.063 ms），畳み込みを中心とする CAE は演算密度が比較的高く，Tensor Core を活用した最適化の恩恵を受けやすいと考えられる．一方で，Stacked RNN 部では FP16 が FP32 よりわずかに遅くなる傾向が確認された（RNN avg: 17.066 ms → 17.688 ms）．本モデルの RNN 部は Foresight モジュールにより 1 推論周期内で複数回の RNN 推論を実行するため，小サイズの行列積が多数発生する．実際に，図 2a–2c で示す各 RNN モジュール内の行列サイズは最大でも batch size が 5 のときの Slow RNN の隠れ状態であり，5×60 程度である．このような小サイズ行列では，FP16 で Tensor Core を用いる際のタイルサイズに起因するパディングにより，実際のテンソルより大きいタイルで演算してしまうため無駄な計算が発生する．また TensorRT の Auto-Tuner が最適なカーネルを選択する際，FP32 では小行列向けに最適化されたカーネルが選択される一方で，FP16 では同等のカーネルが選択されない可能性がある．さらに，型変

表 2: 推論遅延の比較（単位: ms）

環境	CAE avg	CAE max	RNN avg	RNN max
Desktop PyTorch CPU	1.740	4.881	19.433	73.954
Desktop PyTorch GPU	0.720	1.248	24.198	43.205
Jetson PyTorch GPU	8.674	18.016	101.237	132.153
Jetson TensorRT (FP32)	1.075	1.392	17.066	23.055
Jetson TensorRT (FP16)	1.063	1.610	17.688	20.639

表 3: デスクトップ環境での電力計測結果（単位: W）

backend	CPU core	CPU package	GPU	Total
PyTorch CPU	19.879	21.632	21.955	43.587
PyTorch GPU	16.995	18.728	47.216	65.944
アイドル時	2.017	3.721	21.817	25.538

表 4: Jetson Orin Nano 環境での電力計測結果（単位: W）

backend	CPU_GPU_CV	SOC	VDD_IN
PyTorch	2.198	2.458	8.205
TensorRT FP32	2.005	2.455	7.956
TensorRT FP16	2.002	2.452	7.946
アイドル時	1.627	2.419	7.453

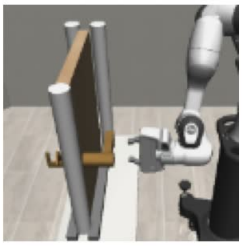
換やバイアス加算・累積を FP32 で行う混合精度のオーバヘッド，および GPU カーネル起動回数の増加が支配的になると，FP16 の理論的な演算性能が遅延低下に直結しない．以上より，本モデルでは CAE は FP16 でわずかに高速化する一方，RNN は演算規模・実行形態の制約により FP16 の利点が現れにくく，わずかに遅延が増加した．

4.2.2 消費電力

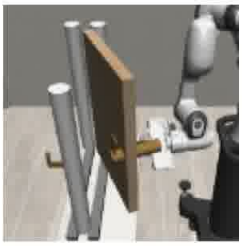
表 3 および表 4 に，それぞれデスクトップと Jetson の電力計測結果を示す．デスクトップ環境では，Intel RAPL を用いて CPU コア，CPU パッケージを計測し，NVML を用いて GPU の消費電力を計測した．システム全体の電力は，CPU パッケージと GPU の和として算出した．Jetson 環境では，内蔵の電力モニタを用いて，VDD_CPU_GPU_CV，VDD_SOC，VDD_IN の 3 つの電源ラインの消費電力を計測した．VDD_IN はシステム全体の電力に近似できるため，これをシステム全体の電力として報告する．

デスクトップ環境と比較して Jetson では低消費電力で実行でき，さらに FP16 では FP32 よりも消費電力が低下する傾向を確認した．

表 4 より，Jetson はアイドル時点で VDD_IN が 7.45 W であり，推論（CAE+RNN）実行時の増分は PyTorch で 0.75 W，TensorRT で 0.50 W 程度と見積もれる．したがって，TensorRT 化は制御周期を満たすための遅延改善だけでなく電力削減にも寄与し，発熱低減やバッテリ駆動時間延長に有効である．一方で，FP16 と FP32 の電力差は小さいため，最終的な精度要件とあわせて，精度低下が許容できる範囲で FP16 を選択するのが現実的である．

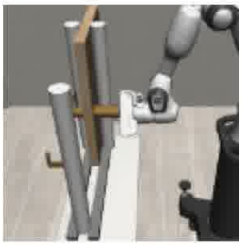


(a) 初期状態



(b) pull-type（成功例）

(c) push-type（成功例）



(d) slide-type（成功例）

図 4: シミュレーション環境の初期状態と成功例

5. シミュレーション評価

5 節では，TensorRT への変換および FP16 適用がタスク性能へ与える影響を，成功率の観点から評価する．実機評価では環境条件のばらつきや再現性の制約があるため，条件を揃えやすいシミュレーション環境で比較を行った．3.2 節で述べたほうに，比較条件は，PyTorch FP32，TensorRT FP32，および CAE を FP32 のまま RNN のみを FP16 で動作させる混合精度構成（TensorRT RNN FP16）である．

本稿のシミュレータタスクでは，ドア種別として pull，push，slide の 3 種類を対象とした．成功の場合はドアを閾値以上開けられた状態とし，失敗の場合はドアの開きが不十分で，ほとんど動かない状態を含むものとした．図 4 に，シミュレーション環境の初期状態と，各ドア種別の成功例を示す．

5.1 成功率評価方法

シミュレータで，1 エピソード 200 ステップの実行において，期間中に一度でもドアが閾値以上開いた場合を成功と定義した．閾値未満の場合を失敗と定義し，ドアが動か

表 5: シミュレーション環境での成功率

backend	pull	slide	push
PyTorch FP32	97	99	39
TensorRT FP32	96	99	16
TensorRT RNN FP16	93	94	53

ない・開き量が不十分なケースを含む．閾値設定は，pull と push type で回転角 0.1 rad，slide type は x 方向に 0.1 m とした．また，UF-RNN モデルは推論時に乱数ノイズを用いるため，random seed を変えて各条件 100 回試行し，成功率を評価した．

5.2 成功率評価結果

各条件 100 回の試行結果から成功率を算出した．表 5 に成功率（%）を示す．

pull-type および slide-type は，いずれの条件でも高い成功率を示しており，Jetson への移植（TensorRT FP32）による性能低下は限定的である．また，TensorRT RNN FP16 でも成功率は高い水準を維持しており，これらのタスクに対しては RNN の FP16 化による影響は相対的に小さい．push-type は元々成功率が低くランダム性が高いため，結果の分散が大きい．TensorRT FP32 で成功率が低下した一方，TensorRT RNN FP16 では成功率が改善している．このことは，数値精度や実装差に起因する誤差が時系列制御の状態遷移へ影響し得ることを示している．

UF-RNN モデルでは，推論時に複数のノイズ系列を評価・選択する構造を持ち，FP16 化による微小な数値誤差が隠れ状態に影響し，ノイズ系列の選択や行動出力を変える．さらに，ロボット制御は，時系列タスクであるため，ある時刻の数値誤差は，RNN 内部状態や環境を通して次時刻以降に伝搬し，数値誤差が累積・増幅する．これらの要因 FP16 化による pull/slide 成功率低下の原因と考えられる．

6. まとめ

本稿では，不確実性駆動型先見予測に基づく UF-RNN モデルを対象として，計算資源と電力制約の厳しいエッジ環境（Jetson Orin Nano）におけるリアルタイム実行可能性と，数値精度選択が推論特性およびタスク成功率へ与える影響を評価した．まず，推論バックエンドを TensorRT へ変換することで推論遅延のばらつきを抑制し，10Hz の制御周期で安定して動作させられることを確認した．また電力計測より，デスクトップ環境と比較して低消費電力なエッジ環境でも同等のタスク実行が可能であることを示した．

深層学習で広く利用されている FP16 精度計算の適用は，消費電力削減効果がある一方で，RNN 部で小行列演算が多数回発生する構成では，推論遅延は必ずしも改善されない．さらに，モデルの特性上，FP16 化に伴う数値誤差が時系列制御タスクの成功率へ影響し得ることを示した．

以上より，先見予測型の時系列制御モデルをエッジ環境へ展開する際には，推論遅延・消費電力に加えて，精度変更に伴う成功率の変化を含めた総合評価に基づき，適切な推論基盤と精度（FP32/混合精度/FP16）を選択することが重要である．今後の課題として，より細かい粒度の混合制度計算の導入に加え，対象ハードウェアの特性（例：Tensor Core 等）を踏まえた行列サイズの選択を含むモデル設計レベルでの最適化，あるいはモデルの行列サイズ特性に適したハードウェア選択が挙げられる．

謝辞 本研究の成果の一部は JST【ムーンショット型研究開発事業】【JPMJMS2031】の支援を受けたものです．

参考文献

[1] Ito, H., Yamamoto, K., Mori, H. and Ogata, T.: Efficient multitask learning with an embodied predictive model for door opening and entry with whole-body control, *Science Robotics*, Vol. 7, No. 65, p. eaax8177 (online), DOI: 10.1126/scirobotics.aax8177 (2022).

[2] Hiruma, H., Ito, H. and Ogata, T.: UF-RNN: Real-Time Adaptive Motion Generation Using Uncertainty-Driven Foresight Prediction, *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 8004–8011 (online), DOI: 10.1109/IROS60139.2025.11245661 (2025).

[3] Hiruma, H., Ito, H. and Ogata, T.: Adaptive Motion Generation Using Uncertainty-Driven Foresight Prediction (2024).

[4] NVIDIA: Jetson Orin Nano Developer Kit, `https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/`. Accessed: 2026-01-12.

[5] NVIDIA: NVIDIA TensorRT Documentation, `https://docs.nvidia.com/deeplearning/tensorrt/`. Accessed: 2026-01-12.

[6] Argall, B. D., Chernova, S., Veloso, M. and Browning, B.: A survey of robot learning from demonstration, *Robotics and Autonomous Systems*, Vol. 57, No. 5, pp. 469–483 (online), DOI: https://doi.org/10.1016/j.robot.2008.10.024 (2009).

[7] Suzuki, K., Ito, H., Yamada, T., Kase, K. and Ogata, T.: Deep Predictive Learning: Motion Learning Concept inspired by Cognitive Robotics (2024).

[8] Hochreiter, S. and Schmidhuber, J.: Long Short-Term Memory, *Neural Computation*, Vol. 9, No. 8, pp. 1735–1780 (online), DOI: 10.1162/neco.1997.9.8.1735 (1997).