ChatGPT

ResNet-BK関連効率化技術の最新研究動向

現代の大規模モデルを効率的に扱うために、**ResNet-BK**プロジェクトでも注目されている様々な最適化技術があります。本レポートでは以下の7つのテーマについて、最新の研究論文・記事や実装例、ベンチマーク結果、直面している課題、学習者向けリソースなどを網羅的にまとめます。

1. ハードウェア最適化（CUDAカーネル最適化、Triton、Tensor Coresなど）

● **CUDAカーネルのカスタム実装と最適化:** 深層学習フレームワークが提供する汎用オペレーションだけでなく、演算を融合した専用CUDAカーネルを自作することでGPUの性能を極限まで引き出せます。例えば**FlashAttention**では通常複数ステップに分かれる注意機構を1つのCUDAカーネルにまとめ、メモリへの中間書き戻しを削減しました [1] [2]。この手法により**FlashAttention**は標準実装より2〜4倍高速になり、さらに改良版の**FlashAttention-2**ではA100 GPU上で最大約230 TFLOPS相当（FP16精度）の計算スループットを達成しています [3]。2024年にはH100に最適化された**FlashAttention-3**が登場し、Tensor Coreを活用して前版より1.5〜2倍速い740 TFLOPS（FP16）に到達しています [4]。このように特定処理向けにカーネルを最適化すれば、GPUの理論性能に迫る「スピード・オブ・ライト」実装も可能になります。

● **OpenAI Tritonによる高水準カーネル記述:** CUDA C++でカーネルを書くのは難易度が高いですが、OpenAIの**Triton**はPythonライクな記法で高性能カーネルを書けるドメイン固有言語です [5]。Tritonを用いるとスレッドブロックやメモリタイリングを抽象的に指定でき、開発者は計算ロジックに集中できます。例えば上記FlashAttentionの実装でも一部にTritonが使われています。また2023年以降のPyTorch 2.x系では、コンパイラ（TorchInductor）がグラフ中の演算を自動的に融合し、Tritonコード生成によってカーネル数を削減する最適化が導入されています。これにより典型的なモデルのEager実行に比べてGPUメモリアクセスが減り、実行速度が向上します。ResNet-BKでも、`bk_core.py`内の主ループをTritonやCUDA C++カーネルに置換する実験タスクが計画されており [6]、Volta/Turing世代のNVIDIA T4 GPUでレイテンシとスループットの改善が期待されています。

● **Tensor Coreの活用:** NVIDIA GPUのTensor Coreは行列演算を高速化する専用ユニットで、FP16やINT8など低精度演算で**混合精度**を用いることで大幅な高速化が可能です [7]。例えばFP32精度のままでは毎秒数十TFLOPS程度の性能しか出ない処理も、FP16にすればTensor Coreで数百TFLOPS級の性能を発揮できます [8]。実際FlashAttention-2ではSoftmax計算など非行列演算部分を削減することで、A100上で理論性能の72%にあたる225 TFLOPSの学習スループットを達成しました [3]。ResNet-BKでもタスク「A-04: Tensor Cores活用検討」にあるように、行列演算へ問題を帰着させる工夫や低精度化によりTensor Coreを積極的に使う戦略が重要です。特にT4のようなGPUでは演算リソースが限られるため、FP16への切り替えやINT8量子化による推論高速化（TensorRTの利用など）が実用上有効でしょう。

● **メモリ階層・共有メモリのチューニング:** ハードウェア最適化では単に計算ユニットの利用率を上げるだけでなく、**メモリアクセス最適化**も不可欠です。GPUではグローバルメモリ(HBM)アクセスはレイテンシ数百サイクルと遅いため、できるだけレジスタやSM内の共有メモリを活用しデータ再利用することが性能向上の鍵となります [9] [10]。たとえばCUDA 12.0以降では**レジスタスピルを共有メモリに逃がす**新機能が追加され、レジスタ不足時の局所メモリ書き込みを減らすことで5〜10%の速度改善が確認されています [11] [12]。またスレッドブロックあたりの共有メモリ使用量とレジスタ使用量のバランス調整も重要で、occupancy（同時並行実行スレッド数）を最大化するために**launch bounds**やループアンローリングの度合いを調整するテクニックが知られています [13] [14]。これらの低レベル最適化は高度なGPUプロファイリングと試行錯誤が必要ですが、NVIDIAの最適化ガイドやサンプルコードが参考になります [15]。ResNet-BKでは初期PoC段階で完全

最適化は行われませんが [16] 、将来的な課題としてCUDAカーネルのレジスタ削減・共有メモリ活用が挙げられています [17] 。

● **実装例とベンチマーク:** ハードウェア最適化の効果は実験によって裏付けられています。先行研究「FlashAttention」のコードはGitHubで公開され広く利用されています [18] 。またMicrosoftの**DeepSpeed**やNVIDIAの**FasterTransformer**ライブラリには、CUDA最適化されたAttentionやTransformer層の実装が含まれており、大規模言語モデル(LLM)の高速化に寄与しています。特に長いシーケンス長での性能差は顕著で、たとえば32kトークン長のGPT-4クラスのモデルでもFlashAttention実装なら現実的な時間で計算可能です [19] 。一方で最適化カーネルはGPUアーキテクチャに強く依存し、コードの保守・移植が難しいという課題もあります。T4など比較的古いGPUではAmpere/Ada向けの最新手法がそのまま使えない場合もあるため、各世代GPUに応じたチューニングが求められます。教育リソースとしては、公式のCUDAベストプラクティスガイド [20] やTritonのチュートリアル、そして実際の高速化ライブラリのソースコード解析が大学生にも勉強になるでしょう。

2. 勾配チェックポイントとPyTorchのメモリ節約技術

● **勾配チェックポイント (Activation Checkpointing) の仕組み:** 勾配チェックポイントとは、**順伝播の中間活性を一部保存せずに都度再計算する**ことでメモリ使用量を削減するテクニックです [21] [22] 。標準的なニューラルネットワークの訓練では各層の出力を保持し逆伝播で使いますが、大量の中間テンソルがメモリを圧迫します。Checkpointingを適用したブロックでは順伝播出力を保持せず、逆伝播時にそのブロックの順伝播をもう一度実行して勾配計算に利用します [23] [24] 。これによりメモリフットプリントを大幅に削減でき、**計算量とのトレードオフ**でより大きなモデルやバッチサイズを扱えるようになります [21] [25] 。PyTorchでは `torch.utils.checkpoint.checkpoint` 関数で任意の順伝播関数を包むだけでこの挙動を実現でき、ResNet-BKの `bk_core.py` や `resnet_bk.py` 内でもブロックをcheckpointで包む実験が提案されています [26] 。コード例としては次のように、モデルの一部を関数化してcheckpointに渡します。

```
import torch
from torch.utils.checkpoint import checkpoint

# 一例: モデルの順伝播一部をcheckpointする関数
def forward(self, x):
    x = checkpoint(self.block1, x)  # block1の出力を保持せず再計算
    x = checkpoint(self.block2, x)  # block2も同様にチェックポイント
    x = self.block3(x)              # smallなblock3はそのまま
    return x
```

このように大きなブロック（メモリ使用量の多い部分）のみチェックポイントするのが一般的です [27] [28] 。ResNet-BKでもまずBKコアを含むブロックをcheckpoint化し、ミニバッチを倍増してもGPUメモリに収まるかを検証するタスクが設定されています [29] 。

● **メモリ削減の効果とオーバーヘッド:** Checkpointingにより**必要メモリは最大で約1/2〜1/3**に削減できます [30] （極端な場合、保存しない場合と全保存の場合の中間程度）。例えばあるモデルでバッチサイズ16が限界だったのが、チェックポイント適用後はバッチサイズ64まで拡大できた、という報告もあります。ただしデメリットとして**逆伝播計算が遅くなる**点に注意が必要です [31] 。各チェックポイント部分で順伝播を再実行するコストが加算されるため、訓練全体が平均で数割程度遅くなります。またPyTorchでの実装上、極端に細かく分割しすぎると却ってオーバーヘッドが増えメモリ節約効果も薄れることが知られています [32] 。実際、チェックポイントを増やしすぎるとかえってメモリ消費が増えるケースも報告されており [32] [33] 、適切な粒度で配置することが重要です。最新研究では、この**最適なチェックポイント配置を自動探索**する手法も提案されています。例えばPyTorch開発者コミュニティでは、計算グラフを点集合に見立ててmin-cut問題に落と

し込み、グラフカットにより保存すべきノード集合を決定するアルゴリズムが議論されています [34] [35]。この手法では勾配計算に必須なテンソルのみを保存し、それ以外は再計算するスケジュールを自動構築することで、**理論的に最小限のメモリでオーバーヘッドも抑える**ことに成功しています [34] [30]。研究プロトタイプではResNet50などで従来より25%高速かつメモリ節約を両立する結果も報告されています [36]。

● **PyTorchにおける他のメモリ節約技術:** 勾配チェックポイント以外にも、PyTorchや周辺ツールには様々なメモリ節約策があります。例えば**静的グラフ最適化**により不要なテンソルを早期フリーする手法、勾配そのものを保持せず都度計算（逆モード自動微分ではなく順伝播を繰り返す）する極端な方法も研究されています。また**Gradient Offloading**といって、アクティベーションや勾配テンソルを一時的にCPUホストメモリやディスクに退避する実装も一部のフレームワーク（DeepSpeedなど）で導入されています。DeepSpeedのZeRO技術ではオプティマイザ状態や勾配を複数GPUに分散保持することで各GPUのメモリ負荷を下げる工夫もあります。単GPU環境でも、例えば**Mixed Precision訓練**（FP16やBF16での訓練）はメモリ削減策の一つです。メモリ帯域節約にもなるため、多少の再現性問題に留意しつつ多くのプロジェクトでAMP(自動混合精度)が標準採用されています。ResNet-BKではPhase5の未実装項目として「勾配チェックポイントとメモリ節約技法」が挙げられており [37]、今後の展開としてDeepSpeedの融合やPyTorch FSDPの適用なども考えられます。

● **制約と課題:** メモリ節約技術はモデル訓練を可能にする上で重要ですが、いくつかの課題もあります。Checkpointingに関しては**再計算による速度低下**と**実装の複雑さ**が問題です。どの層をチェックポイントすべきかはモデルによって異なり、自動探索アルゴリズムは計算コストが高く実用段階ではありません。またOffloadingはデータ転送がボトルネックとなり、場合によっては計算より通信に時間を取られることもあります。Mixed Precisionは数値誤差によるモデル劣化や不安定性を招く可能性があります。例えばLayerNorm周辺での精度劣化や勾配のゼロ化問題が知られており、適切なloss scalingやBF16の活用が必要です。教育的リソースとしては、PyTorch公式の**Activation Checkpointingチュートリアル** [38] やAWSの**SageMaker開発者ガイド** [39] にわかりやすい説明があります。またオープンソース実装（例えばハグフェイスのTrainerで `gradient_checkpointing=True` を有効にしている例など）を読んでみるのも理解の助けになります。大学生であれば、小さなモデルでメモリ使用量を計測しつつcheckpointの有無でどれだけバッチサイズを増やせるか試す実験は、有益な学習経験となるでしょう。

3. 計算削減技術（Adaptive Computation Time、Early Exitなど）

● **Adaptive Computation Time (ACT) の概念: Adaptive Computation Time (適応計算時間)**とは、**入力ごとに動的にニューラルネットの計算量を調整する**手法です [40]。もともとGraves氏らがRNN向けに2016年に提案したもので、各タイムステップで「十分な情報が処理されたか」を判定する**halting unit**を設け、一定基準に達したら計算を打ち切る仕組みです [40]。これをTransformerなどに応用した例として、**Universal Transformer**や**Adaptive-Transformer (Layer Adaptive)**の研究があります [40]。ACTでは未処理の入力に対して追加の反復計算を行い、難易度に応じて処理深度を変えるため、簡単な入力には少ないステップで、高度な入力にはより多くのステップを割り当てることができます [41] [42]。最近では大規模言語モデルの**チェイン・オブ・ソート推論**(CoT)において、不要に長い推論を**モデル自身が途中で停止**できるようにする試みがあり、RLを用いて不確実性に基づきステップ数を制御する研究 [43] [44] や、各ステップで解答の収束を検知して早期終了する手法（Early Stopping CoT [45]）も提案されています。これらはいずれも「必要以上に考えすぎる(overthinking)」問題に対処する一種のACTと言え、**計算資源の節約**と**推論高速化**を狙った動的推論技術です [41] [46]。

● **早期終了 (Early Exit) ネットワーク:** より一般的な**Early Exit**とは、ニューラルネットワークの中間層に複数の**出口（分類器）**を設け、入力がある程度「簡単」であれば途中の出口で最終結果を出力して計算を打ち切る手法です [47]。ディープモデルの各層は入力の抽象度を上げる役割がありますが、簡単な入力なら浅い層の特徴だけで十分判断できることが多いため、無駄な深層計算を省略できます。画像分野では早くから**BranchyNet** [48] や**Shallow-Deep Network**の研究があり、NLP分野でも**BERT**を対象にした「Patience-Based Early Exit (BERT Loses Patience)」 [49] や**DeepBayes**など多数の提案があります。特に「BERT早期終

了」に関するNeurIPS 2020論文BERT Loses Patienceでは、**一定回数連続して高信頼な予測**が得られたら残り層をスキップする戦略を導入し、平均計算コストを大幅削減しました [49] [50]。この方法では各Transformer層後に追加の分類ヘッドを置き、推論時に例えば「直近3層で予測がほぼ変化しなければ終了」といった基準で動的に終了します [49]。実装例として、この論文の公式コードPABEEではPyTorchで中間出力にソフトマックスを適用し、閾値を超えたら `break` する処理が記述されています [50] [51]。他にも**FastBERT** [52] や**DEE-BERT**、**Multi-Exit ViT**など早期退出ネットワークのバリエーションが提案されています。

● **Token単位の動的プルーニング:** 近年は**トークンごとの動的計算**にも注目が集まっています。通常のEarly Exitはシーケンス全体で一律に終了する（**シーケンス単位の早期終了**）ことが多いですが、より細粒度には**トークンごとに異なる深さまで処理する**アプローチも考案されています [53]。例えば**Hash routing**を用いたものや、各トークンにスコアを割り振り重要度の低いトークンの計算を途中で省略する**Dynamic Token Pruning**があります [54] [55]。TR-BERT [56] や**PoWER-BERT**（ICLR 2020）ではSelf-Attentionで自己重要度の低いトークンを層が深くなる前に間引くことで計算量とメモリ使用を削減しました。2021年の**Magic Pyramid**という研究では、**層方向のEarly Exit**と**トークン方向のプルーニング**を組み合わせ、入力によっては浅い層＋一部トークンのみで予測し、高難度入力では深い層まで全トークンを処理するピラミッド型の適応モデルを提案しています [57] [58]。このようなマルチスケール適応手法により、BERTベースのモデルで最大50%以上の計算削減を実現しつつ精度低下を数%以内に抑えた報告もあります [48] [59]。

● **実装例とベンチマーク:** Early Exitの実装は比較的シンプルで、既存モデルに数層おきにclassifierを追加訓練すればよいため、学術界だけでなく工業界でも応用されています。例えばMicrosoftの学習型Edge AIでは端末上でDNNを走らせる際に早期退出ネットワークで推論を高速化する研究があり、Exit予測モジュールで「難しい入力」を見分けて計算をバイパスする工夫をしています [48] [60]。この研究ではExitの判断に軽量な予測器を用い、従来中間分類器を各層に入れていた場合よりオーバーヘッドを減らしています [48] [59]。ResNet-BKにおいてもタスクC-01「Early-Exit ResNet-BKブロックの設計」が挙げられており [61]、BKコアブロックの途中に早期終了機構を組み込む構想があります。もし実現すれば、ある入力ではBKブロックを1回だけ適用して次に進み、複雑な入力では複数回繰り返すといった動的計算が可能になるでしょう。ベンチマークとしては、BERT系列ではGLUEタスクで推論レイテンシが平均30〜40%短縮された例 [51] や、ResNet系でもCIFAR-10で計算量を半減しつつ精度ほぼ維持などの結果が文献にあります [47]。ただしDynamicなモデルはハードウェア実行効率との兼ね合いもあり、実際のディープラーニングフレームワーク上では分岐によるSIMD効率低下など課題もあります。近年の研究では**スリマブルネットワーク**（幅も動的変更可能なネット）とEarly Exitを組み合わせて**SLEXNet**のような動的幅・深さネットを提案する動きもあります [62]。

● **課題と展望:** 適応計算手法の課題は、**訓練の難しさ**と**評価の複雑さ**にあります。Early Exit付きモデルは各出口で十分な精度を出すようマルチタスク学習する必要があり、蒸留や特殊な損失設計を要します。また入力ごとに計算量が変わるため、最悪ケースのレイテンシが読みにくく、リアルタイムシステムで採用する際は注意が必要です。さらにモデルの挙動解析も複雑になります。一方で、大規模モデルの推論コスト削減には不可欠な方向であり、将来的には**モデルが自身で計算予算を調節する**自己認識的な推論が一般化する可能性があります。教育用途には、BERT早期終了の論文 [50] やサーベイ論文 [63] が参考になります。またGitHub上に**awesome-adaptive-computation** [64] という文献リストが公開されており、様々な手法へのリンクがまとまっています。大学レベルでは小規模ネットで早期終了の実験を行い、例えば難易度別の入力データセットを用意して浅い層での精度と計算削減率を測定すると、動的計算の効果を実感できるでしょう。

4. スパース化（MoEの最適化、BKコアの構造化剪定、N:Mスパース化など）

● **Mixture-of-Experts (MoE) のスパース活性化: Mixture-of-Experts**はモデル内に複数の「エキスパート」層を用意し、**入力ごとにごく一部のエキスパートのみを有効化**することで計算をスパース化する手法です。代表的な**Switch Transformer**では1層あたり数千個のFFNエキスパートを持ち、トークンごとに上位1〜2個のエキスパートだけ計算します [65]。これによりモデルパラメータ総数は大きく増やしつつ、各トークンあたり

の計算は従来と同程度に抑えられます。ResNet-BKでも**MoE層**が実装済みで、top-kルーティングによりエキスパート出力を選択しています [66]。MoEの課題は**ルーターの最適化**と**負荷分散**です。Softmaxで確率的にエキスパートを選ぶ際、一部のエキスパートにトークンが集中すると並列計算資源が無駄になります [67]。GoogleのGShard論文ではこの対策に**Noisy Top-2 gating**と**均等化損失**を導入しました。一方2022年の**Expert Choice**では逆に「エキスパート側からトークンを選ぶ」路由を提案し、負荷不均衡を解消しています [68]。ResNet-BKのタスクD-01「MoEルーティング温度制御」でも、ルータのSoftmaxに温度τを設けてエキスパート選択の分散を調整する実験が計画されています [69] [70]。具体的にはτを上げてSoftmax分布をフラットにすれば負荷分散は改善しますが、逆に専門性は薄れるため精度とのトレードオフになります。最新の研究例では**Hash Layer**（固定のハッシュ関数でトークンを専門家に割当てる）や、**Sparse MoE訓練の勾配近似** [71] などがあり、後者（MicrosoftのSparseMixer）はルータの勾配から省かれる項を補正して学習収束を良くすることで、より高性能なMoEを実現しています [71]。実装面では、Microsoftの**Tutel**ライブラリ [72] やDeepSpeed-MoE [73] がPyTorch向けに高速なMoEカーネルを提供しており、大規模GPU環境で数千億パラメータのMoEモデルを訓練する実績があります。MoEは特に言語モデルで一時注目されましたが、Vision分野やマルチモーダルにも応用が広がっており、例えば2024年のMoE-LLaVAでは視覚質問応答モデルにMoEを組み込み、専門家分岐で幻覚を減らす効果を報告しています [74]。

● **構造化剪定 (Structured Pruning): モデル剪定**は不要な結合やニューロンを削減する圧縮手法ですが、特に**構造化剪定**はチャネル単位やヘッド単位でユニット丸ごとを削減する方式です。ResNet-BKでも**BKコア内部の構造化剪定**が100倍圧縮の一環として試験導入されたとされています [37] [75]。構造化剪定の利点はハードウェア上で速度向上につながりやすい点です。例えばCNNならフィルタ（チャネル）を削除すればその層の演算が直接減り、ライブラリも対応しやすいです。一方、**非構造（ランダム）剪定**は重み行列内の個別要素を零にしますが、疎行列演算は密行列ほど効率良く実行できず、50〜90%重みをゼロにしても実測ではさほど速度が向上しない場合があります [76] [77]。そこで、なるべく計算効率を損なわない**パターン化スパースネス**が注目されています。NVIDIA Ampere世代がサポートした**2:4構造的スパース性**はその代表例で、**どの4つの重みでもうち2つだけ非ゼロ**というパターンを全層で適用します [78]。この制約下ではハードウェア（Sparse Tensor Core）がゼロ演算をスキップでき、理論上**2倍のスループット**が得られます [79]。実際A100 GPUでは2:4スパース対応のcuBLASが用意されており、対応モデルでは約1.5〜1.8倍の速度向上が報告されています [79]。ストレージ面でも2:4ならメタデータは2bit/値と効率的で、16bit重みの場合約44%のメモリ削減になります [80]。ResNet-BKではINT4量子化や剪定を組み合わせ100×圧縮を達成したとありますが [37] [75]、その中にはこのような構造スパース化も含まれていると推測されます。

● **N:Mスパース化の訓練と最適化:** N:Mスパース（2:4はN=2,M=4の一例）は硬直なパターンゆえに訓練が難しい面があります。ゼロにする重みの組合せは組み合わせ爆発的に多いため、効率的なアルゴリズムが必要です。2021年の研究では、勾配を用いて各グループ内のN個の非ゼロインデックスを学習する**逐次最適化法**が提案され、TransformerやResNetで精度維持に成功しています [81]。またN:MスパースはTransformerの自己注意を高速化する用途にも使われ、例えばMishraらの研究ではGPT-2の事前学習を2:4スパースで行い、スループット向上と学習安定化の両立を示しました [79]。NVIDIAは公式に**SparseGPT**などのツールを提供していませんが、サードパーティの実装としてはMITの**SCRevive**やAlibabaの**Structured Sparsity Learning**などがあります。構造化剪定全般の課題として、**精度劣化の評価**と**ファインチューニング**があります。単純なL1ノルム基準でチャネル剪定すると精度が大きく低下することもあり、剪定後に蒸留で回復させたり、ラグランジュ乗数法で最適なバランスを探す研究もあります。またTransformerでは**マルチヘッド注意のヘッド剪定**も盛んに研究され、不要なAttentionヘッドを削除しても精度影響が小さいことが報告されています（例えばMichelらのACL2019論文）。ResNet-BKのタスクG-04「構造化剪定アルゴリズムの比較」でも、様々な剪定手法（正則化による徐々剪定や、一発剪定後の再訓練、AutoMLによる最適構造探索など）の効果をベンチマークする計画があります [82] [83]。

● **スパースモデルの高速化ツール:** 圧縮後のスパースモデルを高速に動かすには専用ライブラリが必要です。NVIDIAのcuSPARSE [84] やCUTLASSは疎行列計算をサポートしており、PyTorchでも `torch.sparse` モジュールがあります。しかし一般の非構造疎行列は並列処理が難しく、GPUでは密行列に比べ性能が出にくいです [76]。そこで、構造的な疎性（N:Mやブロック疎）を活かせる**Sparse Tensor Core**対応のAPIが近年整備され

ました。またCPU向けには**DeepSparse**エンジン（Neural Magic社）が90%超スパースモデルをx86上で高速実行するなどの事例もあります。Mixture-of-Expertsの場合、推論時に未使用エキスパートを計算しない実装（DeepSpeed-MoEやFastMoE）が必須です。近年の研究では**MoEモデルのオンデマンドロード**も議論されており、2023年のOffloading MoE論文では使用頻度の低いエキスパートをディスクに退避しLRUキャッシュで読み込む仕組みを提案しています [85]。この手法では直前のトークンで使われたエキスパートは次も使われやすい性質を利用し、エキスパートロードを推測することで**大規模MoEを単GPUでも扱える**ようにしています [85]。一方、極端な圧縮として**QMoE**ではMoEの各重みを1ビット以下相当にまで圧縮（20倍圧縮）し、1兆パラメータ相当のモデルを一般GPUで載せる試みもあります [86]。このようにスパース化技術は多岐にわたり発展していますが、課題として**疎性に適した学習率調整**や**ハードウェア依存性**などがあります。今後はソフトウェアとハードの連携（例えば将来GPUでN:M以外のパターンもサポートする等）や、疎モデルの評価指標（スパース度合だけでなくエネルギー効率など）の確立も求められるでしょう。

5. マルチスケール処理・トークン縮退（周波数埋め込み、階層的BKコアなど）

● **マルチスケール処理の必要性:** 従来Transformerなどでは全入力トークンに対し同じ系列長・同じ解像度で計算を行います。しかし、入力には局所的パターンと大域的パターンが混在し、**異なるスケールで情報を処理**する方が効率的な場合があります。**マルチスケール処理**とは、モデル内に異なる時間・空間スケールの表現を組み込むことです。視覚分野ではCNNがプーリングで解像度を落としつつ高レベル特徴を捉えるのが典型例であり、Vision Transformerでも**SwIN Transformer**のようにパッチサイズを変化させ階層的特徴マップを作るアプローチがあります。一方、言語や系列データでも**階層的な構造**が内在するため、マルチスケール処理が注目されています。ResNet-BKの文脈では、**BKコアを階層的に構成**し長い系列でも計算量線形で高表現力を保つことが狙われています [87] [88]。タスクE-02「マルチスケールBKコア階層の構築」はまさにそれを目指したテーマです [89] [90]。具体的なアイデアとしては、まず入力系列を複数の解像度（例えば元の細粒度トークン列と要約された粗粒度トークン列）に分け、それぞれBKコアで処理した後に融合する、といった構造が考えられます。あるいはBKコア自体にピラミッド型の演算（例えば粗い時間ステップで大局的変動を捉え、細かいステップで微細構造を詰める）を組み込むかもしれません。

● **周波数埋め込み (Frequency Embedding): 周波数埋め込み**とは、入力信号に対して異なる周波数成分を明示的に符号化してモデルに与える手法です。機械学習においては座標へのサイン波埋め込み（Positional Encodingの一種）やFourier特徴がよく使われます。NeRFなどでは位置に複数周波数のSin/Cosを適用して高周波情報をMLPで扱いやすくしました。同様に自然言語や系列データでも、高速変動成分と低速トレンド成分を分離して学習させることで効率化が期待できます。Googleの**FNet**はTransformerの自己注意の代わりに**全埋め込みに対し一括でFFT**を適用する型破りなモデルでしたが、BERTに匹敵する精度を出しつつ計算を大幅に高速化しました [91]。FNetは学習可能重みを持たないフーリエ変換層でトークン同士の混合を行うことで**擬似的に大域的相互作用**を実現しています [91]。結果としてGLUEベンチマークでBERTの92-97%の精度を達成しつつ、学習は約80%高速（GPU）・70%高速（TPU）だったと報告されています [91]。この成功例は、周波数領域での情報表現が自己注意の近似になり得ること、そして計算量を削減できる可能性を示しています。また2023年の研究**StreamingLLMs with Attention Sinks**では、あらかじめ定めた位置に「グローバル情報を貯めるトークン（sink）」を設置し、各トークンがウィンドウ内局所注意＋そのsinkへの注意のみ行うという**定常スパース注意**を導入しています [92] [93]。これも一種の周波数分割的アプローチで、すべての情報を細粒度にやり取りするのでなく一部を集約して扱う手法です。ResNet-BKのタスクE-01「マルチスケール周波数埋め込みの導入」では、入力埋め込みに対しWavelet変換やFFT的な処理を組み込むことが検討されている可能性があります [94]。例えば原時系列をそのままBKコアに入れるのでなく、低周波成分と高周波成分に分けて別々に処理し再合成する、といった構想です。この場合、低周波部分は粗く処理しても支障なく、高周波部分は局所的に処理すればよいので、全体の計算量削減につながるでしょう。

● **トークン縮退 (Token Merging/Pruning):** 長い系列入力に対しては、途中で**トークン数を減らしていく**工夫もマルチスケール的発想です。視覚分野では画像パッチを段階的にマージする**Token Merging (ToMe)** [95]や、不要パッチを落とす**ViT用プルーニング**が2022〜2023年に数多く提案されました。同様に言語でも、例

えば文脈中で冗長となったトークンをマスク・削除していく**Length Adaptation**があります。2023年の研究Dynamic Token Pruning for LLMsでは、自己注意のアテンションスコアに基づき**重要度の低いトークンを動的に間引く**方法を提案し、長文入力で推論を高速化しています [96]。また**MrT5 (ICLR 2025)**ではバイトレベルのT5モデルにおいて、隣接するトークンをマージしていくことで推論コストを削減する手法が示されています [97]。一連の手法は概ね、浅い層では細かいトークン分解で局所情報を取得し、深い層では全体像だけ残すという**ピラミッド型処理**を実現しています [57] [98]。ResNet-BKにおいてBKコアはO(N)計算ですが、さらにトークン数N自体を減らせれば理論上は入力長に対してサブリニアなスケーリングも期待できます。

● **階層的モデル構造:** マルチスケールを活かすには**階層型アーキテクチャ**が有力です。言語モデルでは昔から**階層型RNN**（文脈を文や段落で階層表現する）や**段落ごとにTransformer**を用いる試みがありました。近年、長文入力に対しまずチャンクごとにエンコードし、その後チャンク表現をまとめて再度エンコードする**Hierarchical Transformer**が提案されています。また**Longformer**や**BigBird**のように決まったスパースな注意パターンで局所・大域の2段階注意を行う手法もマルチスケールの一種です [99]。ResNet-BKのBKコアが解いている漸化式は本来全長の依存を持つものですが、もしこれを多段に組み替え、小さなブロックで近隣計算→圧縮→さらに計算という構造にできれば、大幅な計算削減につながるでしょう。タスクE-02ではそのような「BKコアの階層」を構築するとあり [89]、大変チャレンジングですが達成できれば画期的です。参考になる文献として、階層型ネットの成功例に**Hierarchical Vision Transformer**や前述のHierarchical Transformer (段落Transformer)があります。またWaveNetの文脈では**マルチレゾリューションCNN**で高レートと低レートの2系列を同時生成する手法もありました。教育リソースとしては少々高度ですが、NeurIPS 2021のMagic Pyramid論文 [58] が読めればマルチスケール＋早期終了の考え方がよく理解できます。またGitHubのAwesome Token Reductionリポジトリ [100] に多数の関連論文が整理されています。今後の展望として、マルチスケール処理は長距離依存を低コストで扱う鍵技術となる可能性があります。ResNet-BKの開発においてもこれを取り入れることで、より長いシーケンスや高周波成分を効率よくモデリングできるようになるでしょう。

6. 学習理論（Koopman作用素による学習、GRAD_BLENDの理論解析など）

● **Koopman作用素とディープラーニング: Koopman作用素**は力学系の非線形な挙動を**高次元の線形作用素**として表現する数学理論です [101] [102]。非線形システムの状態xに対し、ある関数空間上の線形作用素$\mathcal{K}$が存在し、$\mathcal{K} \phi(x) = \phi(f(x))$（$f$は系の遷移関数）を満たす基底関数$\phi$の組を見つけられれば、非線形動態を線形で解析できます。しかし実際にはKoopman固有関数ϕを解析的に求めるのは困難であり、**ディープラーニングでこの固有関数を近似**しようという研究が2010年代後半から盛んです [103] [104]。代表的なものにLuschらのNature論文「Deep learning for universal linear embeddings of nonlinear dynamics」があり、オートエンコーダ構造で潜在空間にKoopman固有基底を学習させています [101] [104]。彼らのネットワークは力学系データから**非線形座標変換**を学習し、その空間ではシステムが線形に進行することを示しました [103]。このアプローチにより、従来困難だった**長期予測**や**制御**が線形理論で扱えるようになる可能性が示唆されています [101] [105]。ResNet-BKでKoopmanが話題に出るのは、BKコアが**解析解可能な線形作用素**（三重対角行列の逆の対角要素計算）を含むためでしょう。BKコアの理論的背景にKoopman作用素や固有関数の考え方があり、タスクF-01「Koopman作用素の基礎理論調査」 [106] とF-02「Koopman固有関数を学習するニューラルアプローチ」 [107] が設定されています。具体的には、BKコアが解いている漸化式をKoopman理論で捉え直すことで、新たな学習則や安定性の理論解析につなげる狙いがあると考えられます。例えば**Koopman固有値**が訓練済みモデルの時系列予測誤差と関係するか、固有モードを直接制御して学習させる方法がないか、といった研究方向が考えられます。関連文献としては、2018年以降多くの「Deep Koopman」論文があり [108] [109]、例えばLiらのICLR2020論文ではVAEでKoopman空間を学習し高精度予測を実現しました。また近年は**Physics-Informed Koopman**など物理法則を組込んだ学習も検討されています [110]。理論面ではKoopman作用素のスペクトル解析や連続スペクトルへの対処など難題も残りますが、深層学習と組み合わせることで実用的に解を得る動きが広がっています [102] [105]。ResNet-BKにおいてKoopman的視点を取り入れると、モデルの挙動を固有モード分解して理解しやすくなる可能性があります。

● **GRAD_BLENDの概念と理論解析: GRAD_BLEND**はResNet-BK独自のハイブリッド勾配手法で、**解析的な勾配**と**数値的（自動微分の）勾配**を適度にブレンドして学習する仕組みです [111] [112]。BKコア部分に関しては数式的に導出可能な勾配（おそらく三対角行列逆問題の微分）が存在し、それを組み込みつつ通常のバックプロパゲーションも併用することで、勾配計算の安定性や精度を向上させています [111]。ResNet-BKレポートによれば、仮説7に基づく解析勾配と自動微分勾配の混合比（grad_blendパラメータ）をグリッドサーチした結果、Transformerベースラインを大幅に上回るパープレキシティ改善が得られたとのことです [113] [114]。具体的にはシーケンス長512でPerplexity 428と報告され、これはbaseline Transformerより良好です [113]。なぜ勾配をブレンドすると性能が上がるのか、これを解明するのが「GRAD_BLENDの理論解析」にあたります。考えられる理由として、解析勾配は数値微分では捉えにくい長距離依存の正確な勾配成分を提供し、一方で自動微分勾配はモデル全体の複雑な誤差信号を含むので、両者を組み合わせることで勾配情報がリッチになる可能性があります。また解析勾配のみでは過学習やロバスト性に問題がある場合、ノイズ的な自動微分勾配を足すことで汎化性能が向上する、といった見方もできます。類似の発想として、物理インフォームドNNで**解析解に対する残差をNNで学習**させるケースがありますが、GRAD_BLENDは勾配版と言えます。理論解析では、この手法が勾配降下法に与える影響や収束性を評価することになるでしょう。もし数理的に定式化できれば、「勾配ブレンドはある目的関数の自然勾配法に対応する」などの理解が進むかもしれません。現段階では先行研究がほとんど無い新規性の高い試みですので、ResNet-BKチーム内の検証結果や直感に基づき仮説検証していくことになるでしょう。

● **その他の理論的検討:** 学習理論の観点では、ResNet-BKが従来のTransformerと異なる収束挙動やスケーリング則を示す可能性があり、その分析もテーマになり得ます。例えばStep2で報告された**Perplexityの大幅改善** [113] がなぜ起きたのか、理論的にはBKコアの持つ帰納的バイアス（Inductive Bias）が寄与しているのでは、と推察できます。Koopman理論的に言えば、BKコアが特定のスペクトル性質を持つため長期依存を捉えやすい、といった説明が考えられます。さらに、Phase2で試みられた解析勾配と数値勾配のハイブリッドは、一種の**勾配修正**手法とも見做せ、他のモデルへの応用可能性や一般解釈も気になります。理論解析には高度な数学が伴うかもしれませんが、関連コミュニティでは**ニューラルODE**や**連続力学系としてのResNet**などの研究も進んでおり、そうした枠組みにBKコアを位置付けて議論すると理解が深まりそうです。教育リソースとしては、Koopman理論の入門にはBrunton博士らの教材や上記Nature Communications論文 [101] [104] が役立ちます。また「ディープラーニングと物理法則」を扱った記事や、Koopman＋NNのレビュー論文も参照すると良いでしょう。Grad_blendに関しては前例が少ないため、関連しそうな**解析的勾配の利点**を論じた論文（例えば物理シミュレーションで誤差逆伝播 vs 解析微分を比較した研究など）を調べる価値があります。大学生には難しいテーマですが、逆に**未知の問題を理論的に探究する面白さ**を体験できる場でもあるでしょう。

7. モデル圧縮手法（量子化、蒸留、構造化剪定、評価指標）

● **量子化 (Quantization):** モデル圧縮でもっとも直接的なのが**量子化**です。重みやアクティベーションを低ビット表現にすることでモデルサイズと演算量を削減します [115]。ポストトレーニング量子化(PTQ)と量子化対応学習(QAT)の2種類があり、PTQは**学習済みモデルを追加学習なしで量子化**する手法、QATは**量子化を考慮して再訓練**する手法です [116] [117]。PTQは手軽ですが精度劣化のリスクがあり、特にLLMのように繊細なモデルでは工夫が必要です。近年、LLM向けに**重みのみ量子化 (weight-only PTQ)**が盛んで、GPTQという手法が登場しました [118]。**GPTQ**はレイヤーごとに重みを最適量子化し誤差を補正するアルゴリズムで、GPT-3クラスの巨大モデルでも4ビットや3ビット量子化を高精度に実現したと報告されています [118]。例えばLLaMA-65Bを4bitにしてもほぼ性能維持でき、メモリ1/8に圧縮できるという衝撃的な結果が2023年に示されました。またAWQ [119] やSpQR [120] など改良手法も提案され、重要な重みは高精度保持・他は低精度化といった工夫で2bitまで挑戦するケースもあります [121]。一方**重み＋アクティベーション量子化**も進んでおり、ZeroQuantやSmoothQuantはLLMの全演算をINT8に収める試みに成功しています [122]。 [123] ZeroQuantはグループ単位量子化で重みを、SmoothQuantはチャネルスケーリングで活性の外れ値を抑えることで、精度維持したまま8bit化しています [124]。ResNet-BKでもPhase4で8bit整数化や複素量子化が試されています [66] [75]。特に「複素数量子化」は珍しいキーワードですが、おそらくBKコアが複素スペクトルを扱うため、重

みを実数部・虚数部に分けて量子化するアプローチかもしれません。さらにINT4 MoEという記述から、エキスパート部分も4bit化したようです [66]。

● **ナレッジ蒸留 (Knowledge Distillation):** 蒸留は**大きなモデル(教師)の知識を小さなモデル(生徒)に移す**ことで後者の性能を高める技術です。モデルサイズ圧縮の古典的手法で、DistilBERT [75] やTinyBERTのように、BERT-baseを1/2や1/4のパラメータに縮小しても95%以上の精度を維持する成功例があります。ResNet-BKでもG-02タスクで蒸留フレームワーク実装が計画され、Transformer教師のsoftターゲットをResNet-BK生徒に学習させるとあります [125]。蒸留では教師モデルの出力確率分布や中間層表現を損失に組み込むことで、生徒モデルが単独学習よりも高性能になるよう誘導します。LLM分野でも、大規模モデルの対話データ出力を蒸留して小型チャットボットを作る例（Stanford Alpacaなど）が登場しました。蒸留の利点は専用ハード不要で**精度劣化が少ない圧縮**が期待できる点ですが、欠点は**再学習に時間がかかる**点です。教師並みの性能まで引き上げるには生徒モデルに十分な容量と訓練データが必要で、結果としては4倍圧縮程度が限界というケースもあります。ResNet-BKでは既に蒸留でperplexity比較実験が計画されており [126]、おそらくTransformerとResNet-BK間でモデルの表現力差を埋める試みがなされるでしょう。これは学術的にも興味深く、異種アーキテクチャ間の知識移植という意味で貴重なケーススタディになります。

● **モデル剪定:** 剪定については前述の構造化剪定(テーマ4)と重複しますが、モデル圧縮全般の文脈では**非構造化剪定**（重みマスク）も含め議論されます。非構造化剪定は精度への影響が比較的少なく高い剪定率を達成しやすいですが、疎演算による実速度向上が限定的です。一方構造化剪定は速度向上確実ですが、わずかな剪定でも精度低下が大きい場合があります。最近はその中間として**ブロック単位剪定**や**Low-Rank近似**も活用されています [127]。Low-Rank近似は大きな重み行列をランクrの2つの小行列積に分解する手法で、GPT-2サイズのTransformerで約2倍高速化できた例もあります。ただしランク削減による表現力低下を補うため追加学習が必要です。ResNet-BKでは剪定アルゴリズム比較タスクがあり、そこで評価指標として**モデルサイズ・速度・精度**のトレードオフを見積もると思われます [82]。例えば各手法で圧縮率90%達成時の精度を比較する、といった検証が考えられます。

● **モデル圧縮の評価指標:** 圧縮技術の効果を測る指標としては、多くの場合**圧縮率(パラメータ削減率)**と**スピードアップ(推論時間短縮率)**、そして**精度低下**の三者をバランス見る必要があります [115] [128]。場合によっては**エネルギー消費**や**メモリ使用量**も重要です。例えば組み込みAIではモデルサイズ(MB)やRAM消費が制約となるため、それを指標にします。LLMではGPU数やVRAM容量がボトルネックになるため、FP16で350GB必要なGPT-3を8bitにして175GB、4bitで90GBと削減し単一サーバで扱えるようにする、といった評価が現実的です [129] [130]。また**タスク性能の保持率**も指標となります（例: 圧縮後も元モデルの90%精度を維持）。ResNet-BK文脈では、おそらく**perplexityの劣化幅**や**速度向上倍率**を評価するでしょう。READMEによれば100×圧縮でなお学習能力を示したとの記載があるため [37]、そこではパープレキシティがどの程度維持されたかが重要です。さらに長期的には、モデル圧縮による**環境負荷軽減**や**コスト削減**といったマクロな指標も考慮されます。モデル圧縮の教育的資料としては、近年のサーベイ「A Survey on Model Compression for LLMs」 [131] が総合的で、中でも量子化・蒸留・剪定・低ランク分解の章立てが参考になります [115] [132]。また実践面では、TensorFlow LiteやOpenVINO、TorchTensorRTなど圧縮モデルを実行するプラットフォームのドキュメントも現場感覚を養うのに役立ちます。大学生には、自分で例えばMNISTのMLPを8bit量子化してみる、Distillによって小型モデルを訓練してみる、といったハンズオンが理解を深めるでしょう。

● **現在の制約と今後:** モデル圧縮技術は目覚ましい発展を遂げていますが、なお課題もあります。極端な低ビット量子化（例えば2bit以下）は未だ汎用的とは言えず、特に生成系モデルで品質劣化が問題です。蒸留は教師モデルの挙動すべてを伝えることは難しく、知識欠落（例えば特定モーダルの性能低下）が起こりえます。剪定は訓練と一体化しないと高圧縮率で性能維持は難しく、Lottery Ticket仮説など理論研究も続いています。評価指標も単純な精度以外に、**公平性やロバスト性**への影響を測る必要が提起されています。例えば圧縮でバイアスが増幅されないか、敵対的耐性が下がらないか、といった点です。今後はモデル圧縮と**蒸留・微調整**を組み合わせた包括的な最適化が進むと考えられます [133]（例: MoE + 蒸留 + 量子化の統合）。ResNet-BKプロジェクトでもPhase4でこれらを総合したパイプラインに言及しており [66]、非常に先進的な

取り組みといえます。圧縮技術は、計算資源が限られる研究者や企業にとって今後一層重要になるでしょう。そのための理論的支柱と実装ツールの両面で、今回挙げた技術群はこれからも発展していくはずです。

参考文献: 本レポートでは各節に具体的な論文や記事の参照（【】で示した）を付してあります。興味があるテーマについて、ぜひ元の文献にも当たって詳細を確認してください。それぞれ最新の知見が詰まっており、深く学ぶことで効率的なモデル設計と実装の理解が深まるでしょう。

6 16 49 50 11 12 76 92 40 91 113 66 101 104 118 134

---

1 2 8 9 10 76 77 84 92 93 99 A Meticulous Guide to Advances in Deep Learning Efficiency over the Years | Alex L. Zhang
https://alexzhang13.github.io/blog/2024/efficient-dl/

3 18 19 FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning · Hazy Research
https://hazyresearch.stanford.edu/blog/2023-07-17-flash2

4 Next Generation of FlashAttention | NVIDIA Technical Blog
https://developer.nvidia.com/blog/next-generation-of-flashattention/

5 Simplifying CUDA kernels with Triton: A Pythonic Approach to GPU Programming | by Arun Jith A | Medium
https://arunjitha.medium.com/simplifying-cuda-kernels-with-triton-a-pythonic-approach-to-gpu-programming-79bb7121e974

6 16 17 26 29 37 49 61 66 69 70 75 82 83 87 88 89 90 94 106 107 111 112 113 114 125 126 ResNet-BK 研究タスク.pdf
file://file_000000002d3c720e9d7f3163529a4194

7 Deep Learning Software (JA-JP) - NVIDIA Developer
https://developer.nvidia.com/ja-jp/deep-learning-software

11 12 13 14 How to Improve CUDA Kernel Performance with Shared Memory Register Spilling | NVIDIA Technical Blog
https://developer.nvidia.com/blog/how-to-improve-cuda-kernel-performance-with-shared-memory-register-spilling/

15 20 Mastering CUDA Kernel Development: A Comprehensive Guide
https://medium.com/@omkarpast/mastering-cuda-kernel-development-a-comprehensive-guide-1f3032666b94

21 22 23 24 25 27 28 31 32 33 Mastering Gradient Checkpoints in PyTorch: A Comprehensive Guide | Python-bloggers
https://python-bloggers.com/2024/09/mastering-gradient-checkpoints-in-pytorch-a-comprehensive-guide/

30 34 35 36 Min-cut optimal(*) recomputation (i.e. activation checkpointing) with AOTAutograd - compiler - PyTorch Developer Mailing List
https://dev-discuss.pytorch.org/t/min-cut-optimal-recomputation-i-e-activation-checkpointing-with-aotautograd/467

38 Activation Checkpointing - Amazon SageMaker AI
https://docs.aws.amazon.com/sagemaker/latest/dg/model-parallel-extended-features-pytorch-activation-checkpointing.html

39 How to save memory by fusing the optimizer step into the backward ...
https://docs.pytorch.org/tutorials/intermediate/optimizer_step_in_backward_tutorial.html

40 41 42 43 44 45 46 openreview.net
https://openreview.net/pdf?id=0WdN7pFCja

47 52 64 67 68 71 74 85 86 133 koayon/awesome-adaptive-computation - GitHub
https://github.com/koayon/awesome-adaptive-computation

48 59 60 リソース制約下の早期終了予測を用いたエッジAI (Resource-Constrained Edge AI with Early Exit Prediction) | AI Busi
https://aibr.jp/2025/03/28/%E3%83%AA%E3%82%BD%E3%83%BC%E3%82%B9%E5%88%B6%E7%B4%84%E4%B8%8B%E3%81%AE%E6%97%A9%E6%9C%9F%resourc/

50 51 GitHub - JetRunner/PABEE: Code for the paper "BERT Loses Patience
https://github.com/JetRunner/PABEE

53 [PDF] ConsistentEE: A Consistent and Hardness-Guided Early Exiting ...
https://ojs.aaai.org/index.php/AAAI/article/view/29922/31612

54 57 Magic Pyramid: Accelerating Inference with Early Exiting and Token ...
https://neurips.cc/virtual/2021/34282

55 [PDF] Dynamic Token Pruning for Efficient Long Context LLM Inference
https://openreview.net/pdf?id=gGZD1dsJqZ

56 [PDF] TR-BERT: Dynamic Token Reduction for Accelerating BERT Inference
https://aclanthology.org/2021.naacl-main.463.pdf

58 98 [PDF] Magic Pyramid: Accelerating Inference with Early Exiting and Token ...
https://neurips2021-nlp.github.io/papers/21/CameraReady/magic_pyramid.pdf

62 Adaptive Inference Using Slimmable Early Exit Neural Networks
https://dl.acm.org/doi/10.1145/3689632

63 Early-Exit Deep Neural Network - A Comprehensive Survey
https://dl.acm.org/doi/full/10.1145/3698767

65 72 Tutel: An efficient mixture-of-experts implementation for large DNN ...
https://www.microsoft.com/en-us/research/blog/tutel-an-efficient-mixture-of-experts-implementation-for-large-dnn-model-training/

73 [PDF] DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and ...
https://proceedings.mlr.press/v162/rajbhandari22a/rajbhandari22a.pdf

78 80 arxiv.org
https://arxiv.org/pdf/2104.08378

79 Accelerating Transformer Pre-Training with 2:4 Sparsity - arXiv
https://arxiv.org/html/2404.01847v1

81 [PDF] LEARNING N:M FINE-GRAINED STRUCTURED SPARSE NEURAL ...
https://openreview.net/pdf?id=K9bw7vqp_s

91 [2105.03824] FNet: Mixing Tokens with Fourier Transforms
https://arxiv.org/abs/2105.03824

95 96 97 100 GitHub - ZLKong/Awesome-Collection-Token-Reduction: A collection of token reduction (token pruning, merging, clustering, etc.) techniques for ML/AI
https://github.com/ZLKong/Awesome-Collection-Token-Reduction

101 102 103 104 105 Deep learning for universal linear embeddings of nonlinear dynamics | Nature Communications
https://www.nature.com/articles/s41467-018-07210-0?error=cookies_not_supported&code=8446d3ba-c91d-4106-a06d-b5471acc7207

108 [PDF] Deep learning for universal linear embeddings of nonlinear dynamics
https://www.lri.fr/~gcharpia/deeppractice/2022/chap_5_biblio/Koopman/KoopmanAE.pdf

109 Koopman analysis of nonlinear systems with a neural network ...
https://ui.adsabs.harvard.edu/abs/2022CoTPh..74i5604L/abstract

110 [2211.09419] Physics-Informed Koopman Network - arXiv
https://arxiv.org/abs/2211.09419

115 116 117 118 119 120 121 122 123 124 128 129 130 131 132 134 A Comprehensive Review: Model Compression for Large Language Models (LLMs) | by Doil Kim | Medium
https://medium.com/@kimdoil1211/a-comprehensive-review-model-compression-for-large-language-models-llms-2c0c106d5dbe

127 [PDF] Efficient Large Language Models: A Survey - OpenReview
https://openreview.net/pdf?id=bsCCJHbO8A