Real-Time ViTPose Deployment via Progressive GPU Integration of DARK Post-Processing

Seon Ho Oh, Sangwook Park
Cyber Security Research Division
Electronics and Telecommunications Research Institute
Daejeon, Republic of Korea
{seonho, ssean}@etri.re.kr

***Abstract*—Real-time human pose estimation for multi-camera systems requires near-millisecond processing, but ViTPose deployment is limited by CPU-based post-processing bottlenecks. Traditional implementations create hybrid architectures where GPU-accelerated inference is followed by CPU-based DARK post-processing, leading to performance degradation and GPU underutilization. This paper proposes a TensorRT optimization framework that systematically migrates DARK operations to GPU through progressive integration strategies: partial integration for balanced workload distribution and full integration for maximum GPU utilization. Our approach treats the entire pipeline as a unified optimization target, implementing GPU-optimized DARK operations using vectorized CUDA kernels. Experimental results demonstrate up to 414.4% improvement over PyTorch baseline, achieving 1909 queries per second (QPS) throughput with 0.87ms latency while maintaining identical AP performance (76.9%) with no accuracy degradation in the primary evaluation metric.**

***Index Terms*—ViTPose, TensorRT optimization, GPU acceleration, DARK post-processing, real-time human pose estimation**

I. Introduction

Real-time human pose estimation is critical for computer vision applications including surveillance systems, human-computer interaction, and behavioral analysis. Multi-camera deployment scenarios require processing dozens to hundreds of video streams simultaneously while maintaining near-millisecond response times for immediate anomaly detection and decision making.

ViTPose [1] demonstrates that plain Vision Transformer [2] architectures achieve state-of-the-art performance in human pose estimation, reaching 81.1% AP on MS COCO test-dev dataset. However, the primary bottleneck in production deployment lies in post-processing operations rather than neural network inference complexity. Traditional implementations perform DARK (Distribution Aware coordinate Representation of Keypoint) [3] post-processing on CPU, creating hybrid architectures where GPU-accelerated inference is followed by CPU-based coordinate refinement and transformation.

This hybrid approach creates performance bottlenecks through CPU-GPU data transfer overhead and underutilizes modern GPU parallel processing capabilities. While TensorRT [4] successfully accelerates neural network inference, CPU-based post-processing implementations leave substantial performance improvements unrealized.

To address these limitations, this paper proposes a TensorRT optimization framework that systematically migrates DARK post-processing operations from CPU to GPU through progressive integration strategies. Our approach treats the entire inference and post-processing pipeline as a unified optimization target, enabling systematic GPU integration that maximizes parallel processing capabilities while minimizing CPU-GPU boundary interactions. The framework employs two integration strategies: partial integration for balanced workload distribution and full integration for maximum GPU utilization, accommodating different deployment requirements and resource constraints.

Our key contributions include: (1) establishing an efficient PyTorch-ONNX-TensorRT conversion pipeline supporting dynamic batch processing, (2) developing progressive integration strategies for systematic GPU migration of DARK post-processing operations, and (3) demonstrating 414.4% performance improvement over PyTorch baseline with 0.87ms latency and 1909 QPS throughput while achieving perfect accuracy preservation (identical 76.9% AP) in the primary pose estimation metric.

II. Related Work

A. Vision Transformers for Human Pose Estimation

The evolution of Transformer-based architectures for human pose estimation has focused on improving accuracy through sophisticated model designs. Early approaches employed complex task-specific architectures, with TokenPose [5] representing keypoints as learnable tokens and TransPose [6] combining Transformers with CNN feature extractors. HRFormer [7] integrated high-resolution representations with multi-scale Transformer blocks to capture fine-grained spatial details.

ViTPose [1] demonstrated that plain Vision Transformer [2] architectures achieve state-of-the-art performance without complex modifications, reaching 81.1% AP on MS COCO test-dev dataset. However, these advances primarily addressed research-level metrics without considering real-time deployment requirements.

B. Coordinate Representation and Post-processing

Accurate coordinate extraction from predicted heatmaps is crucial for pose estimation system performance. Traditional approaches relied on simple maximum detection methods,

which introduced quantization errors and limited sub-pixel accuracy. Huang *et al.* [8] identified systematic biases in coordinate encoding and decoding processes, proposing Unbiased Data Processing (UDP) to address these issues.

DARK [3] advances coordinate decoding by modeling predicted heatmaps as 2D Gaussian distributions through three operations: distribution modulation, sub-pixel refinement via Taylor expansion, and coordinate transformation. While DARK improves coordinate accuracy, its computational complexity creates CPU-based performance bottlenecks that limit real-time deployment in multi-stream scenarios.

C. Deep Learning Inference Optimization

GPU-based inference optimization has achieved success in accelerating neural network computation. TensorRT [4] provides comprehensive optimization through graph-level transformations, including layer fusion, kernel auto-tuning, and precision calibration. TRT-Pose [9] demonstrated TensorRT optimization effectiveness for pose estimation on embedded platforms, achieving real-time performance for single-stream processing. Additional GPU optimization frameworks like ONNX Runtime [10] and OpenVINO [11] have also shown significant acceleration for computer vision models, though primarily focusing on inference rather than integrated post-processing optimization.

However, existing optimization frameworks focus on accelerating model inference while maintaining CPU-based post-processing, creating hybrid systems that fail to fully utilize GPU capabilities. The CPU-GPU data transfer overhead creates system-level bottlenecks that become prominent as inference speeds improve, limiting overall system performance in latency-critical applications.

D. Research Gaps and Motivation

Current literature reveals a disconnect between neural network inference optimization and post-processing acceleration. While research has advanced model inference efficiency through frameworks like TensorRT, post-processing operations rely on CPU-based implementations that create computational bottlenecks, reducing overall system performance by up to 40%.

No prior work has systematically integrated sophisticated post-processing algorithms like DARK into GPU-accelerated inference pipelines. Existing approaches treat inference and post-processing as separate optimization targets, missing opportunities for unified acceleration.

Our work addresses these gaps by developing a framework that treats the entire inference and post-processing pipeline as a unified optimization target. By systematically migrating DARK operations to GPU through progressive integration strategies, we enable substantial performance improvements while maintaining accuracy benefits of sophisticated post-processing algorithms. This integrated approach represents a paradigm shift from component-level optimization to system-level acceleration, addressing critical bottlenecks that limit practical deployment of advanced pose estimation systems.

III. Methodology

A. Optimization Framework

Our framework addresses the computational bottleneck in ViTPose deployment through systematic integration of TensorRT acceleration with GPU-based post-processing. Traditional ViTPose deployment follows hybrid architectures where GPU-accelerated inference is followed by CPU-based post-processing. Our methodology treats the entire pipeline as a unified optimization target, enabling GPU acceleration that reduces CPU-GPU processing boundaries.

B. Model Conversion Pipeline

We establish a PyTorch [12] $\rightarrow$ ONNX [13] $\rightarrow$ TensorRT conversion pipeline optimized for dynamic batch processing. The PyTorch model is exported to ONNX format with dynamic shape support, enabling flexible deployment across varying computational requirements.

The TensorRT conversion incorporates FP16 precision optimization, CUDA graph execution, and layer fusion techniques to maximize GPU utilization. Dynamic batch size configuration supports input shapes from single-image processing (1×3×256×192) to high-throughput batch operations (64×3×256×192).

Validation procedures ensure numerical accuracy preservation throughout the conversion process, with intermediate outputs verified against reference implementations.

C. DARK Algorithm Analysis for GPU Integration

To develop effective GPU integration strategies, we analyze the DARK algorithm from a computational optimization perspective, identifying bottlenecks and parallelization opportunities within each processing stage. This analysis guides our progressive integration approach by revealing which operations benefit most from GPU acceleration and how to structure the migration process for optimal performance.

The DARK method [3] transforms coordinate decoding through three sequential stages: distribution modulation, maximum re-localization, and coordinate transformation. Each stage presents distinct computational characteristics and optimization challenges that must be addressed for successful GPU integration.

1) Distribution Modulation: The first stage applies Gaussian kernel convolution to regularize irregular heatmap distributions:

$$\boldsymbol{h}' = K \circledast \boldsymbol{h} \tag{1}$$

where K represents a Gaussian kernel with standard deviation $\sigma = 2$ and kernel size 11×11.

From a GPU optimization perspective, this convolution operation exhibits high parallelization potential due to its embarrassingly parallel nature across spatial dimensions and multiple keypoints. The separable nature of the Gaussian kernel enables efficient implementation using GPU-optimized separable convolution kernels, significantly reducing computational complexity from $O(n^2)$ to $O(n)$ per spatial dimension. However, the subsequent rescaling operation requires careful

memory management to preserve maximum response magnitudes while maintaining numerical stability.

2) Maximum Re-localization: The second stage performs sub-pixel coordinate refinement to overcome quantization limitations inherent in discrete heatmap representations. The modulated heatmap first undergoes logarithmic transformation to convert the 2D Gaussian distribution into a form suitable for Taylor expansion analysis. The discrete maximum location $\boldsymbol{m} = \arg\max_{(x,y)} \boldsymbol{h}'(x, y)$ is identified within the modulated heatmap as the initial coordinate estimate.

DARK then applies second-order Taylor expansion around this discrete maximum location to achieve sub-pixel precision:

$$\boldsymbol{\mu} = \boldsymbol{m} - (\mathcal{D}''(\boldsymbol{m}))^{-1}\mathcal{D}'(\boldsymbol{m}) \tag{2}$$

where $\mathcal{D}'(\boldsymbol{m})$ and $\mathcal{D}''(\boldsymbol{m})$ represent first and second-order derivatives of the logarithmic heatmap computed using finite differences at the maximum location.

From a GPU optimization perspective, this stage presents the most significant computational bottleneck due to its inherently sequential nature and intensive mathematical operations. The discrete maximum finding operation requires global reduction across spatial dimensions of each keypoint heatmap, which can be efficiently implemented using GPU parallel reduction algorithms. However, the subsequent derivative computation requires irregular memory access patterns around the maximum location that can lead to memory bandwidth limitations on GPU architectures.

The independence of computations across different keypoints enables batch-level parallelization, where multiple keypoints can be processed simultaneously using vectorized operations. The matrix inversion operation in Taylor expansion, while computationally intensive, benefits from GPU tensor core acceleration when implemented using optimized linear algebra libraries. Additionally, finite difference computations can be vectorized across multiple keypoints, allowing simultaneous processing of derivative calculations for improved GPU utilization.

3) Coordinate Transformation: The final stage transforms refined coordinates from heatmap space to original image coordinates, incorporating UDP [8] principles for accurate scaling and offset compensation. This transformation involves primarily element-wise operations and simple arithmetic computations that are well-suited for GPU parallel processing.

The transformation stage exhibits minimal computational complexity but requires careful attention to memory access patterns to avoid performance degradation. Coalesced memory access patterns can be achieved through appropriate data layout optimization, ensuring efficient GPU memory bandwidth utilization.

4) Integration Strategy Implications: The analysis reveals that distribution modulation and coordinate transformation stages are well-suited for immediate GPU acceleration due to their parallel nature and regular memory access patterns. In contrast, the maximum re-localization stage requires more sophisticated optimization techniques, including memory layout optimization and vectorized batch processing.

These insights directly inform our progressive integration strategies: partial integration prioritizes easily parallelizable operations (distribution modulation), while full integration addresses the more challenging re-localization stage through advanced GPU optimization techniques. The computational analysis establishes the foundation for systematic GPU migration that maximizes performance gains while maintaining numerical accuracy.

D. Progressive Integration Strategies

Our approach employs two integration strategies designed to accommodate different deployment requirements and resource constraints.

1) Partial Integration: The partial integration strategy distributes computational workload between GPU and CPU resources to balance performance and system compatibility. Distribution modulation and derivative computation are performed on GPU using separable convolutions. Final coordinate refinement utilizes precomputed derivatives transferred from GPU memory.

This approach provides gradual system integration with minimal architectural changes, balanced resource utilization, and maintains compatibility with existing CPU-based downstream processing.

2) Full Integration: The full integration strategy performs all DARK post-processing operations on GPU, achieving maximum performance through reduced CPU processing overhead. All operations are implemented using optimized CUDA kernels with vectorization across multiple keypoints and dynamic batch sizes.

Key features include vectorized matrix operations utilizing GPU tensor cores, in-place memory operations minimizing allocation overhead, memory coalescing optimizations using aligned memory access patterns, and dynamic batch processing optimization for sizes 1-64.

IV. Experimental Results

A. Experimental Setup

Experiments were conducted on an NVIDIA RTX 4080 GPU using PyTorch [12] 2.7.1, ONNX [13] 1.18.0, and TensorRT [4] 10.3 with FP16 precision optimization. The ViTPose-B model processes 256×192 resolution images, generating 17 keypoint heatmaps corresponding to COCO pose annotation standards.

Performance evaluation employed latency (milliseconds) and throughput (queries per second, QPS) metrics across two optimization profiles: **opt.1** for low-latency applications and **opt.64** for high-throughput scenarios. All measurements were averaged over 100 iterations with 3 warmup iterations to ensure stable performance readings.

B. Accuracy Validation

To ensure GPU optimization does not compromise pose estimation accuracy, we conducted comprehensive accuracy validation using the MS COCO validation dataset [14]. All implementations were evaluated using standard COCO pose

estimation metrics following the official evaluation protocol, including Average Precision (AP) averaged over Object Keypoint Similarity (OKS) thresholds 0.50:0.95 and Average Recall (AR) across multiple OKS thresholds.

Accuracy validation was performed across four implementation variants: (1) PyTorch baseline without TTA (our optimization target), (2) ONNX conversion without TTA, (3) TensorRT full integration without TTA, and (4) PyTorch with Test Time Augmentation (TTA) using horizontal flip averaging (reference for comparison). This validation approach ensures numerical consistency is maintained throughout the optimization pipeline while establishing the baseline performance of our target model.

C. Baseline Performance Analysis

Table I presents baseline performance across PyTorch and TensorRT frameworks. PyTorch baseline achieved 210.08 QPS for single-image processing and 371.10 QPS for batch processing, with 4.76ms latency representing typical research implementation performance.

TensorRT optimization without post-processing demonstrated substantial acceleration, achieving peak performance of 1890.08 QPS for batch processing and single-image performance of 744.38 QPS with 1.34ms latency. This represents a 409.4% improvement in peak throughput and 254.3% improvement in single-image processing over the PyTorch baseline. However, the addition of CPU-based DARK post-processing created significant performance degradation, reducing peak throughput to 1132.68 QPS (40.1% reduction) and single-image performance to 580.59 QPS (22.0% reduction). Latency increased from 1.34ms to 1.72ms, confirming that CPU post-processing is the primary bottleneck limiting real-time deployment.

TABLE I
BASELINE PERFORMANCE ANALYSIS

Method	Opt.	Batch	Latency (ms)	QPS
PyTorch	1	–	4.76±0.20	210.08
	64	–	172.46±1.86	371.10
TensorRT (w/o post-processing)	1	1	1.34±0.07	744.38
	1	64	39.52±0.63	1619.50
	64	1	2.71±0.07	369.27
	64	64	33.86±0.51	1890.08
TensorRT (w/ CPU post-processing)	1	1	1.72±0.09	580.59
	1	64	61.54±0.85	1040.03
	64	1	3.12±0.09	320.10
	64	64	56.50±1.00	1132.68

D. Accuracy Preservation Analysis

Table II presents comprehensive accuracy validation results across all implementation variants. The results demonstrate that our GPU optimization framework maintains numerical accuracy within acceptable tolerances throughout the entire optimization pipeline.

The PyTorch baseline without TTA achieved an AP of 76.9%, establishing our optimization target and reference accuracy for comparison. The ONNX conversion maintained identical accuracy (AP 76.9%) compared to the PyTorch baseline, confirming that model serialization does not introduce numerical errors. TensorRT full integration achieved an AP of 76.9%, demonstrating that our GPU-based DARK implementation preserves coordinate extraction accuracy with **zero degradation in the primary pose estimation metric**.

TABLE II
ACCURACY VALIDATION RESULTS ON MS COCO VALIDATION SET (AP: AVERAGED OVER OKS 0.50:0.95, AR: AVERAGED OVER OKS 0.50:0.95)

Implementation	AP	AP_{50}	AP_{75}	AR
PyTorch (baseline target)	**76.9%**	**90.9%**	**84.2%**	**82.2%**
ONNX Conversion	76.9%	90.4%	84.1%	83.3%
TensorRT Full Integration	76.9%	90.3%	84.0%	83.3%
PyTorch + TTA (reference)	77.5%	91.2%	84.5%	82.6%
Max Accuracy Loss	**0.0% (Perfect)**	**0.6%**	**0.2%**	**-**

Crucially, our optimization achieves perfect accuracy preservation in AP (0.0% loss), the most important evaluation metric for pose estimation systems. Minor variations are observed only in secondary metrics: AP_{50} shows 0.6 percentage points difference and AP_{75} shows 0.2 percentage points difference, which are within typical measurement variance and do not affect the primary performance assessment.

For reference, PyTorch with TTA achieved higher accuracy (AP 77.5%) due to horizontal flip averaging, which provides additional robustness. However, our optimization target is the baseline model without TTA to maintain real-time processing requirements. Notably, both ONNX and TensorRT implementations showed improved Average Recall (83.3% vs 82.2%), indicating that optimized post-processing may provide marginally better keypoint detection sensitivity.

E. Partial Integration Results

Table III demonstrates the effectiveness of strategic GPU-CPU workload distribution. The partial integration approach achieved consistent improvements across all configurations, with QPS gains ranging from 109.85 to 689.08 compared to the CPU post-processing baseline.

Single-image processing improved from 580.59 QPS to 1092.16 QPS (88.1% gain), with latency reducing from 1.72ms to 0.92ms. Notably, this performance exceeded the TensorRT baseline without post-processing (744.38 QPS), indicating that optimized GPU post-processing reduces memory transfer overhead more effectively than eliminating post-processing entirely. The GPU-based convolution operations for distribution modulation leverage optimized CUDA kernels that process multiple keypoints simultaneously, resulting in computational efficiency that compensates for additional post-processing operations.

F. Full Integration Results

Table IV presents complete GPU integration results, demonstrating maximum performance through reduced CPU processing overhead. Peak performance reached 1909.13 QPS, representing a 68.5% improvement over the CPU post-processing baseline and 1.0% improvement over the TensorRT baseline without post-processing.

TABLE III
PARTIAL INTEGRATION PERFORMANCE (GAIN VALUES REPRESENT ABSOLUTE QPS IMPROVEMENT OVER CPU POST-PROCESSING BASELINE)

Opt.	Batch	Latency (ms)	QPS (+Δ)
1	1	0.92±0.08	1092.16 (+511.57)
1	64	39.99±0.64	1600.47 (+560.44)
64	1	2.33±0.09	429.95 (+109.85)
64	64	35.13±0.44	1821.76 (+689.08)

Compared to partial integration, full integration delivers additional performance gains by eliminating all CPU involvement. Peak throughput improved from 1821.76 QPS to 1909.13 QPS (4.8% increase), while minimum latency reduced from 0.92ms to 0.87ms (5.4% reduction). This improvement stems from the complete removal of CPU-GPU synchronization points and maintaining all intermediate data resident in GPU memory throughout the pipeline.

The GPU-based implementation utilizes vectorized operations across multiple keypoints and batch dimensions, enabling parallel processing of Taylor expansion computations and coordinate transformations. Memory coalescing optimizations using aligned memory access patterns ensure efficient access, while in-place operations minimize allocation overhead. These architectural advantages eliminate multiple memory transfer operations: heatmap transfer for coordinate extraction, intermediate result transfers during multi-stage processing, and final coordinate transfers back to GPU memory.

The performance gains demonstrate that treating the entire inference and post-processing pipeline as a unified GPU operation creates combined optimizations that achieve better performance than individual components. By significantly reducing CPU-GPU boundary interactions, full integration achieves optimal resource utilization and maximum throughput for real-time deployment scenarios.

TABLE IV
FULL INTEGRATION PERFORMANCE (GAIN VALUES REPRESENT ABSOLUTE QPS IMPROVEMENT OVER CPU POST-PROCESSING BASELINE)

Opt.	Batch	Latency (ms)	QPS (+Δ)
1	1	0.87±0.08	1150.11 (+569.52)
1	64	39.51±0.66	1619.98 (+579.95)
64	1	2.26±0.09	442.55 (+122.45)
64	64	33.52±0.46	1909.13 (+776.45)

G. Performance Summary

Table V provides a comprehensive comparison of performance improvements across all optimization approaches. Full integration achieved up to 414.4% improvement over PyTorch baseline, with peak performance of 1909.13 QPS and minimum latency of 0.87ms. The progression from PyTorch (371.10 QPS) through TensorRT with CPU post-processing (1132.68 QPS) to full integration (1909.13 QPS) demonstrates 5.14× overall performance improvement.

The experimental results establish three key deployment capabilities: near-millisecond processing (0.87ms) enables real-time applications, high-throughput performance (1909.13 QPS) supports single-GPU processing of multiple concurrent camera streams, and optimized GPU utilization enables deployment in resource-constrained environments **while achieving perfect accuracy preservation in the primary AP metric (0.0% degradation)**.

TABLE V
PERFORMANCE SUMMARY AND COMPARISON

Method	Peak QPS	Min Latency	Improvement
PyTorch	371.10	4.76 ms	–
TensorRT w/ CPU post-processing	1132.68	1.72 ms	+205.1%
Partial Integration	1821.76	0.92 ms	+390.9%
Full Integration	**1909.13**	**0.87 ms**	**+414.4%**

V. CONCLUSION

This paper addressed the critical bottleneck of CPU-based post-processing that limits real-time ViTPose deployment. We proposed a TensorRT optimization framework that systematically migrates DARK post-processing operations to GPU through progressive integration strategies.

Our key contributions include establishing an efficient model conversion pipeline supporting dynamic batch processing, developing progressive integration strategies accommodating diverse deployment requirements, and presenting performance improvements through comprehensive experimental validation. The framework treats the entire inference and post-processing pipeline as a unified optimization target, enabling substantial performance gains through systematic GPU integration.

Experimental results demonstrated up to 414.4% performance improvement over PyTorch baseline, achieving 1909.13 QPS throughput with 0.87ms latency while maintaining numerical accuracy. **Comprehensive accuracy validation confirms perfect preservation of the primary pose estimation metric (AP: 76.9%) with zero degradation**, demonstrating that our GPU optimization maintains pose estimation quality while delivering substantial performance improvements.

The progressive integration approach successfully addresses fundamental limitations of hybrid architectures, transforming computational overhead into performance gains through unified GPU acceleration. These improvements enable practical deployment in demanding real-time applications requiring concurrent processing of multiple video streams with strict latency constraints.

The optimized system supports practical deployment scenarios including near-millisecond processing for interactive applications, single-GPU processing of multiple concurrent camera streams for surveillance systems, and resource-efficient deployment in edge computing environments. The integrated optimization approach demonstrates the importance of treating entire computational pipelines as unified targets rather than optimizing individual components in isolation.

However, the proposed approach has limitations including dependency on specific GPU architectures, potential scalability issues with extremely large batch sizes, and increased GPU

memory requirements that may limit deployment in resource-constrained environments. Future work could explore extending this optimization framework to other vision transformer architectures, investigating integration with tensor processing units (TPUs) and evaluating performance on edge devices with limited GPU memory.

This work establishes integrated pipeline optimization as an effective approach for deploying computer vision models in real-time environments, showing that systematic GPU integration can eliminate computational bottlenecks while maintaining accuracy required for production applications. The demonstrated paradigm shift from component-level to system-level optimization provides a foundation for advancing real-time deployment of sophisticated deep learning models in practical applications.

ACKNOWLEDGMENT

This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (No. RS-2024-00336663, Development of AI technology for tracking and investigating drug criminals through multimedia sources).

REFERENCES

[1] Y. Xu, J. Zhang, Q. Zhang, and D. Tao, "ViTPose: Simple vision transformer baselines for human pose estimation," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022, pp. 38 571–38 584.

[2] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," in *International Conference on Learning Representations (ICLR)*, 2021.

[3] F. Zhang, X. Zhu, H. Dai, M. Ye, and C. Zhu, "Distribution-aware coordinate representation for human pose estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 7093–7102.

[4] NVIDIA Corporation, "NVIDIA TensorRT: Programmable inference accelerator," Software Library, 2023, accessed: 2024-12-19. [Online]. Available: https://developer.nvidia.com/tensorrt

[5] Y. Li, S. Zhang, Z. Wang, S. Yang, W. Yang, S.-T. Xia, and E. Zhou, "Tokenpose: Learning keypoint tokens for human pose estimation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 11 313–11 322.

[6] S. Yang, Z. Quan, M. Nie, and W. Yang, "Transpose: Keypoint localization via transformer," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 11 802–11 812.

[7] Y. Yuan, R. Fu, L. Huang, W. Lin, C. Zhang, X. Chen, and J. Wang, "HRFormer: High-Resolution Transformer for Dense Prediction," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, 2021, pp. 7281–7293.

[8] J. Huang, Z. Zhu, F. Guo, and G. Huang, "The devil is in the details: Delving into unbiased data processing for human pose estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 5700–5709.

[9] NVIDIA Corporation, "Real-time pose estimation accelerated with NVIDIA TensorRT," GitHub Repository, 2023, accessed: 2024-12-19. [Online]. Available: https://github.com/NVIDIA-AI-IOT/trt_pose

[10] ONNX Runtime developers, "ONNX Runtime," https://onnxruntime.ai/, 2021.

[11] OpenVINO Toolkit developers, "OpenVINO Toolkit," https://github.com/openvinotoolkit/openvino, 2018.

[12] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in neural information processing systems*, 2019, pp. 8024–8035.

[13] ONNX developers, "ONNX: Open Neural Network Exchange," https://github.com/onnx/onnx, 2019.

[14] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common Objects in Context," in *European Conference on Computer Vision*. Springer, 2014, pp. 740–755.