Quattro: Transformer-Accelerated Iterative Linear Quadratic Regulator Framework for Fast Trajectory Optimization

Yue Wang[1], Haoyu Wang[1,2] and Zhaoxing Li[1]

***Abstract*— Real-time optimal control remains a fundamental challenge in robotics, especially for nonlinear systems with stringent performance requirements. As one of the representative trajectory optimization algorithms, the iterative Linear Quadratic Regulator (iLQR) faces limitations due to its inherently sequential computational nature, which restricts the efficiency and applicability of real-time control for robotic systems. While existing parallel implementations aim to overcome the above limitations, they typically demand additional computational iterations and high-performance hardware, leading to only modest practical improvements. In this paper, we introduce Quattro, a transformer-accelerated iLQR framework employing an algorithm-hardware co-design strategy to predict intermediate feedback and feedforward matrices. It facilitates effective parallel computations on resource-constrained devices without sacrificing accuracy. Experiments on cart-pole and quadrotor systems show an algorithm-level acceleration of up to 5.3× and 27× per iteration, respectively. When integrated into a Model Predictive Control (MPC) framework, Quattro achieves overall speedups of 2.8× for the cart-pole and 17.8× for the quadrotor compared to the one that applies traditional iLQR. Transformer inference is deployed on FPGA to maximize performance, achieving further up to 20.8× speedup over prevalent embedded CPUs with over 11× power reduction than GPU and low hardware resource overhead.**

I. INTRODUCTION

Model Predictive Control (MPC) is widely used in robotics for trajectory optimization and computes optimal control inputs over future time steps [1], [2], [3]. However, real-time performance in nonlinear settings often demands heavy computational resources [4]. To solve this computationally demanding problem, a prominent family of algorithms is Differential Dynamic Programming (DDP) [5], [6] and its more efficient variant, the iterative Linear Quadratic Regulator (iLQR), which are now widely used in robot control [7], [8], [9]. Despite its strengths, iLQR still relies on a sequential pipeline that heavily demands processor clock frequency [10]. Meanwhile, deep learning (DL) offers massive parallelism [11] and high-performance inference [12], which makes it well suited to address the bottleneck of iLQR. Yet, few works have combined DL with iLQR. This gap highlights an exciting path for faster, more efficient optimal control.

*This work is supported by School of ECS, University of Southampton. The authors acknowledge the use of the IRIDIS High-Performance Computing Facility and associated support services at the University of Southampton in the completion of this work.

[1]Yue Wang, Haoyu Wang and Zhaoxing Li are with the School of Electronic and Computer Science, University of Southampton, United Kingdom. {yue.wang}, {haoyu.wang}, {zhaoxing.li}@soton.ac.uk.

[2]Haoyu Wang is also with the Department of Engineering Science, University of Oxford, United Kingdom.

To overcome the sequential limitation of iLQR, methods like multiple-shooting split the problem into segments [13], [14] for parallel processing using multi-core CPUs or GPUs [15], [16], but introduce extra terms to handle defects between segments [17]. This typically requires more iterations to converge, reducing overall efficiency [10]. More recently, deep learning (DL) methods have emerged as a way to address these challenges. For example, [18] uses a neural network to approximate iLQR outputs and speed up computation, but the network does not capture key sequential dependencies and produces less reliable results. Transformers have also appeared in trajectory optimization, especially in MPC. Celestini et al. apply transformer-based models to generate near-optimal initial guesses, improving convergence and lowering computational cost [19]. Zinage et al. propose TransformerMPC, which predicts inactive constraints and refines initialization [20]. Unlike prior black-box approaches that warm-start generic solvers, our accelerator is tightly integrated with the iLQR algorithm itself, using a Transformer to predict its internal feedback and feedforward gains to directly parallelize its sequential bottleneck for accelerated control.

Deep learning has advanced rapidly over the past two decades, yielding powerful architectures for sequential data. Recurrent Neural Networks (RNNs) [21] and Long Short-Term Memory (LSTM) networks [22] have proven effective in many temporal tasks, but often struggle with vanishing or exploding gradients when modeling long-range dependencies [23]. Transformers [24], [25] avoid these issues by using self-attention, which captures extended context without the strictly sequential updates of RNNs or LSTMs [26]. This design aligns well with the iterative structure of iLQR. A prime example of Transformer's potential in high-temporal tasks is Nvidia's Deep Learning Super Sampling (DLSS), where a transformer-based approach reconstructs high-fidelity gaming frames at higher frame rates [27]. This success directly motivates our pursuit of Transformer-driven iLQR solutions. Moreover, recent advances in deep learning accelerators enable efficient parallelization of Transformer architectures [28], [29], [30], paving the way for real-time, high-performance control applications.

To address the sequential nature of iLQR computations and overcome inefficiencies in existing parallelization methods, we introduce **Quattro**[**] (Figure 1), an iLQR framework that uses a Transformer model to generate the feedback and feedforward terms. By producing these intermediate

[**]Our framework is open-sourced and available at: https://github.com/YueWang996/quattro-transformer-ilqr

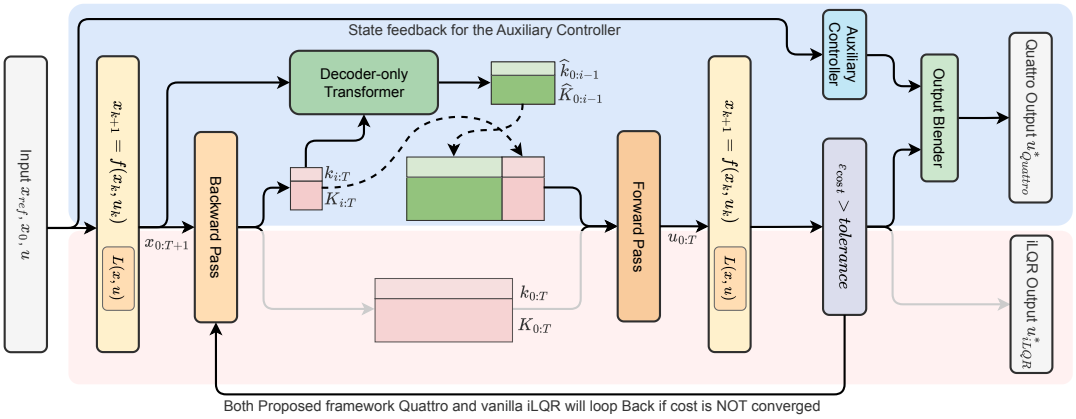

Fig. 1: Comparison of vanilla iLQR (the red-shaded area) and transformer-accelerated iLQR Framework (Quattro, the blue-shaded area).

values in parallel, Quattro reduces computation time while preserving accuracy. We validate the approach on a cart-pole [31] and quadrotor [32] control problems, demonstrating the effectiveness of the framework. Furthermore, an FPGA-based implementation reveals substantial performance gains, underscoring Quattro's potential for real-time applications. The contributions of this paper are as follows:

1) We introduce **Quattro**, a deep acceleration framework for iLQR, intrinsically accelerated by a customized Transformer model. It significantly enhances computational efficiency through parallel computation and immediate inference compensation.
2) We validate the performance of Quattro on cart-pole and quadrotor systems on a variety of computation platforms, achieving remarkable acceleration and comparative accuracy to traditional iLQR.
3) We use the latest accelerator design framework [33] to implement the customized Transformer kernel on FPGA, achieving an optimal balance between performance, power, and hardware overhead. To our knowledge, it is the first deployment of a Transformer model on FPGA specifically for iLQR optimization.

The remainder of this paper is organized as follows: we first introduce the background of the problem in Section II, followed by the details of the proposed framework in Section III. Experiments and results are presented in Section IV, and the paper concludes with Section V.

II. PRELIMINARIES

A. System Dynamics

A system dynamics, or system model, describes how the state of a system changes with a given system input. The model can be defined in a discrete-time differential equation

$$x_{i+1} = f(x_i, u_i), \tag{1}$$

where x_i and u_i represent the state and control input at time step i, respectively. A cost function can be defined to evaluate the performance of the state and input of the system over the next N steps:

$$J(X,U) = \sum_{i=0}^{N-1} l(x_i, u_i) + l_N(x_N), \tag{2}$$

where $l(x_i, u_i)$ is the running cost and $l_N(x_N)$ is the terminal cost. Here, $X := \{x_0, x_1, \cdots, x_N\}$ denotes the state trajectory and $U := \{u_0, u_1, \cdots, u_{N-1}\}$ denotes the input trajectory. We can define an optimal control problem for solving the optimal U and corresponding X such that $J(X,U)$ is minimized:

$$\begin{aligned} \min_{X,U} \quad & J(X,U) \\ \text{s.t.} \quad & x_{i+1} = f(x_i, u_i), \quad \text{given } x_0, U_0. \end{aligned} \tag{3}$$

B. Iterative Linear Quadratic Regulator

The iterative Linear Quadratic Regulator (iLQR) algorithm addresses nonlinear optimal control problems by repeatedly approximating the dynamics and cost around a nominal trajectory. At each iteration, the system dynamics are linearized, and the cost function is approximated up to second order. These local approximations are then used in a backward pass to compute feedback and feedforward control corrections, which subsequently refine the nominal trajectory in a forward pass. Due to the backward pass depending inherently on the results of future steps, the algorithm operates sequentially through the time horizon, inherently limiting parallel implementation. More detailed descriptions of iLQR will be presented in later sections.

C. iLQR in a Model Predictive Control Context

In our work, the iLQR algorithm serves as the core solver within an MPC strategy. This approach works by repeatedly solving a finite-horizon optimization problem at each time step: using the current state of the system as a starting point, iLQR computes an optimal sequence of future control inputs, but only the first input is ever applied. The rest of the sequence is discarded, and the entire process repeats from the new state. This constant replanning allows the controller to continuously correct for trajectory deviations, thereby mitigating errors that arise from inevitable model inaccuracies and external disturbances. The expert solutions

generated by this MPC loop provide the training data for our model, as detailed in Section IV-A.

D. Transformer Architecture and Acceleration

The Transformer architecture, originally developed for sequence modeling in natural language processing [24], [34], relies on stacked encoder-decoder layers with self-attention mechanisms. Unlike recurrent models or sequential algorithms like iLQR, Transformers process all sequence elements simultaneously, enabling efficient parallel computation. Self-attention directly models dependencies across all time steps without recurrence, significantly shortening the information path length and making it easier to capture long-range relationships [24].

This structure makes Transformers well-suited for acceleration on parallel hardware such as GPUs and Tensor Processing Units (TPUs), where matrix operations can be batched efficiently [15]. Applying this architecture to iLQR alleviates much of the overhead from the backward pass that can limit parallelism in the standard dynamic programming approach. By learning to approximate the full control trajectory in iLQR, the transformer-based approach reduces latency and computation time, which is an advantage that is especially beneficial for real-time applications and deployment on edge devices with constrained resources.

The Transformer model is often accelerated on FPGA platforms rather than general computing units, as they are highly customizable and optimized for parallel matrix computations [35]. As one of the Deep Learning (DL) accelerators, numerous novel architectures have been proposed and developed [36], [37], [38], [30], alongside more agile development methodologies [33]. Notably, a new Accelerator Design Language (ADL) named Allo [33] provides a framework to directly translate Python-based transformer models into High-Level Synthesis (HLS) C code with highly optimal latency and hardware overhead. This generated HLS code can then be synthesized into Register-Transfer Level (RTL) circuits for implementation as accelerator Intellectual Property (IP) cores.

III. TRANSFORMER-ACCELERATED ILQR

In this section, we revisit the iLQR computation process and explain how the Transformer enables fast and accurate iLQR computation.

A. System Rollout

The iterative Linear Quadratic Regulator (iLQR) starts by rolling out the nonlinear system dynamics (1) from an initial state x_0 using an initial hypothesis of the control sequence U_0. This produces a nominal trajectory of states X. Using the obtained state-control trajectory, the performance of the trajectory can be measured through the cost function J defined in (2).

B. Backward Pass

The backward pass computes optimal feedback and feed-forward control corrections. First, the nonlinear dynamics is linearized around the nominal trajectory as:

$$\delta x_{i+1} \approx A_i \delta x_i + B_i \delta u_i, \tag{4}$$

where the matrices A_i and B_i are Jacobians of the system dynamics with respect to the state x_i and input u_i.

The cost is approximated to second-order around the same nominal trajectory:

$$\delta l_i \approx \frac{1}{2}\begin{bmatrix}\delta x_i \\ \delta u_i\end{bmatrix}^\top \begin{bmatrix} l_{xx} & l_{xu} \\ l_{ux} & l_{uu}\end{bmatrix}\begin{bmatrix}\delta x_i \\ \delta u_i\end{bmatrix} + \begin{bmatrix} l_x \\ l_u\end{bmatrix}^\top \begin{bmatrix}\delta x_i \\ \delta u_i\end{bmatrix}. \tag{5}$$

Next, following Bellman's principle of optimality, we define a cost-to-go function $V_i(x_i)$:

$$V_i(x_i) = \min_{u_i} \left[l(x_i, u_i) + V_{i+1}(f(x_i, u_i)) \right], \tag{6}$$

with terminal condition $V_N(x_N) := l_N(x_N)$.

By expanding this cost-to-go function locally up to second order, we have:

$$\delta V_i(x_i) = s_i^\top \delta x_i + \frac{1}{2}\delta x_i^\top S_i \delta x_i, \tag{7}$$

where the gradients and Hessians are defined as:

$$s_i = \frac{\partial V_i}{\partial x_i}, \quad S_i = \frac{\partial^2 V_i}{\partial x_i^2}. \tag{8}$$

We similarly expand the state-action cost $Q_i(x_i, u_i)$ as:

$$\delta Q_i = \frac{1}{2}\begin{bmatrix}\delta x_i \\ \delta u_i\end{bmatrix}^\top \begin{bmatrix} Q_{xx} & Q_{xu} \\ Q_{ux} & Q_{uu}\end{bmatrix}\begin{bmatrix}\delta x_i \\ \delta u_i\end{bmatrix} + \begin{bmatrix} Q_x \\ Q_u\end{bmatrix}^\top \begin{bmatrix}\delta x_i \\ \delta u_i\end{bmatrix}, \tag{9}$$

where:

$$Q_x = l_x + A_i^\top s_{i+1}, \tag{10a}$$
$$Q_u = l_u + B_i^\top s_{i+1}, \tag{10b}$$
$$Q_{xx} = l_{xx} + A_i^\top S_{i+1} A_i, \tag{10c}$$
$$Q_{uu} = l_{uu} + B_i^\top S_{i+1} B_i, \tag{10d}$$
$$Q_{ux} = l_{ux} + B_i^\top S_{i+1} A_i = Q_{xu}^\top. \tag{10e}$$

Minimizing the state-action cost with respect to the input variation δu_i yields the optimal control corrections:

$$\frac{d\delta Q_i}{du_i} = Q_u + Q_{ux}\delta x_i + Q_{uu}\delta u_i = 0, \tag{11}$$

giving the optimal solution:

$$\delta u_i^* = k_i + K_i \delta x_i, \tag{12}$$

where:

$$k_i = -Q_{uu}^{-1} Q_u, \quad K_i = -Q_{uu}^{-1} Q_{ux}. \tag{13}$$

The backward pass computes these values recursively from the terminal state to the initial state. Additionally, the gradients and Hessians of the cost-to-go are updated recursively as:

$$s_i = Q_x + K_i^\top Q_{uu} k_i + K_i^\top Q_u + Q_{ux}^\top k_i, \tag{14a}$$
$$S_i = Q_{xx} + K_i^\top Q_{uu} K_i + K_i^\top Q_{ux} + Q_{ux}^\top K_i. \tag{14b}$$

C. Forward Pass

In the forward pass, the control sequence is updated using a line search with step size α:

$$u_i^{new} = u_i + \alpha k_i + K_i(x_i^{new} - x_i), \tag{15}$$

where x_i^{new} is the newly rolled-out state during this forward pass. This produces a new trajectory and updated cost J^{new}.

After the forward pass, convergence is checked by comparing the change in the cost function. If the improvement is smaller than a pre-specified threshold (e.g., $|J^{new} - J| < \varepsilon$), the iteration stops. Otherwise, the procedure repeats the backward and forward passes until convergence or a preset maximum iteration limit is reached. In practical implementations, several step sizes may be tested in parallel: each candidate produces a new control sequence and a forward pass is carried out for each, thereby preventing repeated forward-pass computations if a particular step size proves unsuccessful [15].

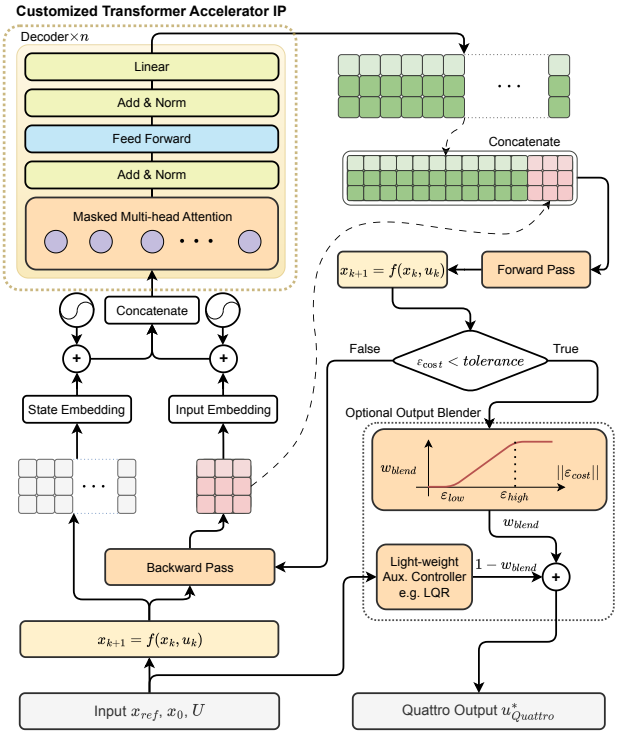

Fig. 2: Detailed framework architecture of the Quattro.

D. Integration of Transformer

The backward pass recursively computes the gains and updates the system input. The core idea to accelerate the algorithm is to reduce the sequential computation. In iLQR, the recursive backward pass is most time-consuming due to the extensive derivatives of system dynamics and cost functions [10].

One idea is to replace the entire backward pass with a Transformer module π_θ as in

$$\hat{k}, \hat{K} = \pi_\theta(X). \tag{16}$$

While converting the entire sequential backward pass to a parallel Transformer inference is much faster, this end-to-end approach can be less robust because it ignores crucial second-order information from the cost-to-go function. Incorporating explicit backward pass information (e.g., cost-to-go Hessians or gradients) as inputs to the Transformer can enable more informed and reliable predictions.

Therefore, in our framework (as Figure 2 shows), instead of computing the full backward pass, we perform a partial backward pass which computes the gain matrices $k_{i:T-1} = \{k_i, \cdots, k_{T-1}\}$ and $K_{i:T-1} = \{K_i, \cdots, K_{T-1}\}$. The rest of the gain matrices are predicted by the Transformer model:

$$\hat{k}_{0:i-1}, \hat{K}_{0,i-1} = \pi_\theta(X, k_{i:T-1}, K_{i:T-1}). \tag{17}$$

By doing this, it preserves the essential structure of optimal control thus improving the robustness and stability of the overall solution.

As in Figure 2, a decoder-only Transformer is adopted for predicting iLQR gain matrices due to its causal structure, which naturally aligns with the sequential and temporal characteristics inherent to gain computations in optimal control. An encoder-based Transformer is less suitable because its bidirectional attention structure does not preserve temporal causality, which is crucial for sequential control tasks. Similarly, a full encoder-decoder Transformer introduces unnecessary complexity and computational overhead without clear advantages, as the primary task here is modeling sequential dependencies rather than mapping between distinct input-output sequences. Thus, the decoder-only Transformer provides a structurally more appropriate choice for modeling the causal and sequential nature of the iLQR gain prediction.

In order to feed both state and gain matrices, we add an additional channel to the Transformer module. The gain matrices are stacked as a single input matrix. After the state embedding and gain embedding, they are positionally encoded and concatenated before being fed to the multi-head attention. The output of the Transformer is a flattened vector which is reshaped to extract the gain matrices $\hat{k}_{0:i-1}$ and $\hat{K}_{0,i-1}$. After concatenating the predicted gains and the calculated gains, the full feedback and feedforward gains are obtained and used to compute the forward pass.

E. Optional Output Blender

To avoid potential oscillations caused by small residual gains from the Quattro near equilibrium, we incorporate an *optional* lightweight Linear Quadratic Regulator (LQR) as an auxiliary controller around the reference state. A standard blending mechanism smoothly transitions between the Quattro and LQR outputs based on the current cost J. Defining two thresholds (ε_{low}, $\varepsilon_{\text{high}}$), the blending weight is computed as:

$$w_{\text{blend}} = \begin{cases} 0, & \|J\| \le \varepsilon_{\text{low}}, \\ \dfrac{\|J\| - \varepsilon_{\text{low}}}{\varepsilon_{\text{high}} - \varepsilon_{\text{low}}}, & \varepsilon_{\text{low}} < \|J\| < \varepsilon_{\text{high}}, \\ 1, & \|J\| \ge \varepsilon_{\text{high}}. \end{cases} \tag{18}$$

The final blended control output is then:

$$u^*_{\text{Quattro}} = w_{\text{blend}}\, u_{\text{predict}} + (1 - w_{\text{blend}})\, u_{\text{LQR}}, \tag{19}$$

which implies that the system is fully controlled by the Transform-predicted result when $w_{\text{blend}} = 1$, and entirely by LQR when $w_{\text{blend}} = 0$.

IV. EXPERIMENTS AND ANALYSIS

We evaluated the Quattro on two benchmark control problems: a cart-pole system and a quadrotor, both simulated using MuJoCo [39].

A. Data Collection

Training data was generated by running the standard iLQR solver within an MPC framework, beginning from a diverse range of initial states. For each MPC control step, we collected the full set of intermediate variables: state trajectories X, control inputs U, and gain matrices (k, K) from each iteration of the underlying solver. This provided a rich dataset of solver behavior across a wide range of system states.

For the low-dimensional cart-pole system $\left(x \in \mathbb{R}^4\right)$, we discretized the dynamics at a 0.01 s time step and simulated for 15 s. Initial x-positions and angles were sampled on a grid over $[-0.5, 0.5]$ in 0.05 increments.

In contrast, for the higher-dimensional quadrotor system $\left(x \in \mathbb{R}^{12}\right)$, grid search is impractical. Instead, we employed Latin Hypercube Sampling (LHS) [40] to draw 2,000 initial states. These spanned $\text{pos}_x, \text{pos}_y \in [-0.3, 0.3]$, $\text{pos}_z \in [0.2, 0.5]$, $\text{roll}, \text{pitch} \in [-0.2, 0.2]$, and $\text{yaw} \in [-0.5, 0.5]$, with all velocities initially set to zero. This ensures broad coverage of the higher-dimensional state space while restricting the initial velocities. We used the same simulation approach to collect iLQR computation results.

B. Transformer Model and Accelerator Design

Transformer models were implemented using PyTorch and trained using the high-performance computing cluster. To address the differing complexities between the two control systems, model parameters were adjusted accordingly. In the case of the cart-pole system, a three-layer decoder-only architecture was employed, with each layer incorporating 4 attention heads and a model dimensionality of 128. The dimension of the feedforward layer is 256, which captures the features of the input sequence. For the quadrotor system, we choose the same Transformer architecture and parameters. However, we extend the feedforward layer from 256 to 512, to capture more input features and dependencies due to the complicated dynamics.

We primarily adopted the recent agile Transformer ADL framework proposed in [33] to build an efficient and practical hardware design. This approach brings together several essential kernel components that form the full Transformer computation. By taking advantage of the parallel matrix multiplication capability of FPGAs, we focused the hardware acceleration on the main computation blocks, including **multi-head attention, layer normalization, residual connections, and the feed-forward network.** These are the most computationally intensive parts of the decoder. Other operations, such as embedding, positional encoding, and matrix combination, were handled by the CPU since they offer limited opportunities for parallel processing. To balance latency and hardware resource usage, we applied several HLS directives such as pipeline, unroll, partition, and the use of dual-port RAM during the design process.

TABLE I: FPGA Development Experiment Setup

Item	Specification
FPGA Device Series	AMD Kintex Series FPGA
Accelerator Design Language	Allo [33]
Main Programming Languages	Python, HLS C, Verilog HDL
EDA Tools	Vitis, Vivado
Clock Frequency	80 MHz (Cart-Pole) 200 MHz (Quadrotor)

C. Prediction Accuracy

For the cart-pole system, we regulate the cart to stop at position 0 along the x-axis and stabilize the rod angle at 0 radians. Figure 3 shows the predicted gain matrices $\hat{k}$ and $\hat{K} = \{K_1, K_2, K_3, K_4\}$ closely matching the actual gains computed from iLQR. Each element K_i corresponds to a feedback gain matrix K component at a given time step.

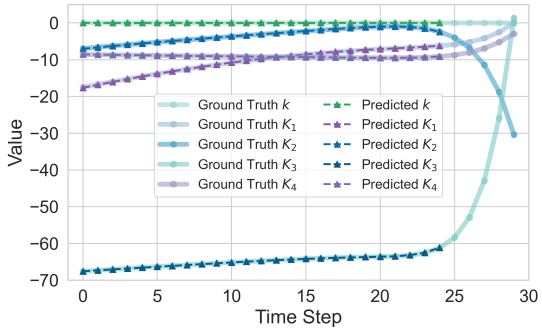

Fig. 3: Comparison between ground truth (solid lines) and transformer-predicted (dashed lines) gain sequences for the cart-pole system, using a prompt length of 5 prior gain steps.

We further evaluated prediction accuracy with different prediction lengths of the intermediate gain matrices. For a specific horizon, iLQR performed a partially backward pass as Section III-D described, and the transformer model predicted the rest of the gain sequence to obtain the updated input sequence $U_{Quattro}$ by a forward pass. We measured the MSE between the $U_{Quattro}$ and the full iLQR computational result. The dotted plot in Figure 4 presents MSE for different payload allocations between iLQR and Transformer. For the cart-pole system ($T = 30$), the MSE values for each sample are relatively centralized and gradually increase as the number of computed iLQR gain matrices decreases.

For the quadrotor system, with a prediction horizon $T = 50$, the Transformer inputs consist of state sequences $X \in \mathbb{R}^{T\times 12}$ and stacked gain matrices $K_{stacked} \in \mathbb{R}^{T\times 52}$ (reshaped from $k \in \mathbb{R}^{T\times 4}$ and $K \in \mathbb{R}^{T\times 4\times 12}$). Figure 4 (lower plot) illustrates MSE, showing higher and more dispersed values due to the increased complexity and dimensionality compared to the cart-pole system. However, despite the higher dispersion resulting from the squared error amplification

across larger gain matrices, most prediction errors remain very low (between 10^{-1} and 10^{-3}). Additionally, for varying prediction lengths, the distribution of MSE samples remains consistent, indicating that a higher proportion of transformer-predicted data can be utilized to maximize parallelization and improve overall computational efficiency.

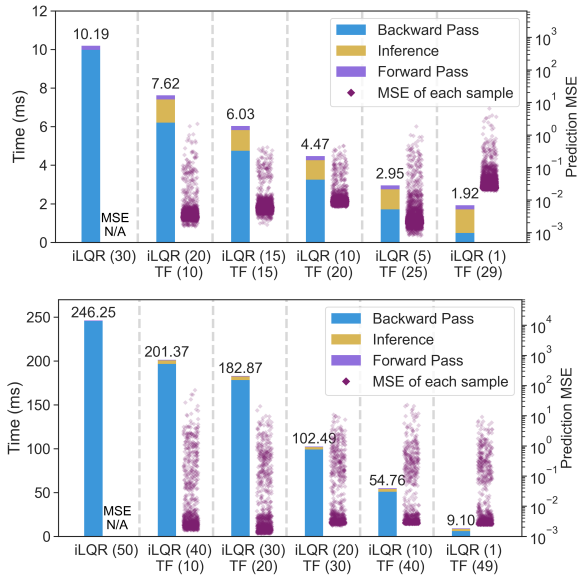

Fig. 4: Running time and prediction accuracy for different computational distributions between iLQR and Transformer for the cart-pole (upper) and the quadrotor (lower) systems. Bar plots show computation time per iteration (left Y axis), and scatter plots indicate MSE distributions (right Y axis) for varying prompt lengths. Numbers in parentheses (e.g., iLQR (30) TF (20)) denote the computational steps performed by iLQR and the Transformer, respectively.

D. Computation Speedup and Accelerator Performance

We first tested our framework on an Apple Silicon M4 Pro (10-core CPU). Figure 4 illustrates computation times for varying allocations between iLQR and the Transformer. As we reduce the computational steps executed by iLQR, overall execution time notably decreases, leveraging the highly parallel computation of Transformer. For the cart-pole system, average computation per iteration reduces from 10.19 ms to 1.92 ms **(5.3× faster)**. For the more complex quadrotor, iteration time decreases significantly from 246.25 ms to 9.10 ms **(27× faster)**, highlighting our method's advantage in handling larger, computationally intensive systems.

Compared to another accelerated iLQR algorithm in [10], which typically requires additional iterations to converge, our method achieves a comparable number of iterations as vanilla iLQR to reach the optimal solution, thereby demonstrating significant overall time savings. As depicted in the upper plot of Figure 5, the total simulation time for the cart-pole system is reduced from 10.5 s to 3.7 s when using 5 iLQR steps and 25 Transformer-assisted steps, resulting in a **2.8× speedup**. This aligns closely with the observed time savings illustrated in Figure 4 (upper). In the quadrotor control scenario, for a simulation with 10,000 total simulation steps and 500 MPC control computations, our method substantially reduces the computation time from 237 s (pure iLQR computation at every MPC step) to 13.3 s by combining 1 step of iLQR computation with 49 subsequent steps predicted by the Transformer. This leads to a **17.8× speedup**. Additionally, when compared to state-of-the-art OCP solvers OSQP, ECOS, and SCS implemented in [41], our approach achieves speedups of 2.49×, 3.63×, and 2.4×, respectively.

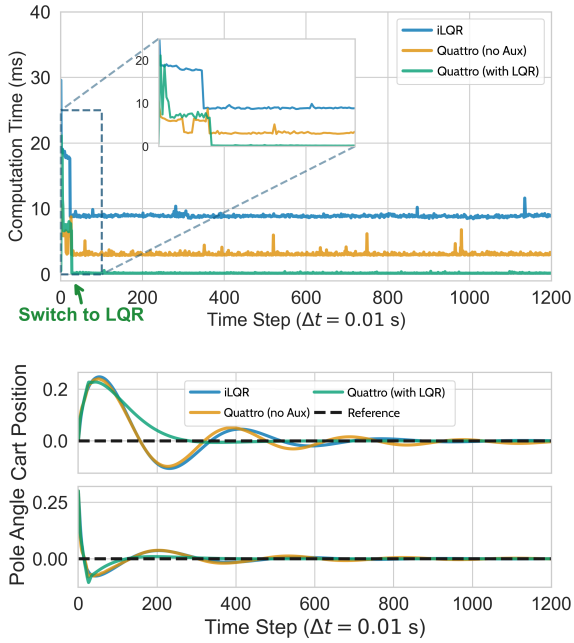

Fig. 5: Cart-pole system: (upper) computational efficiency and (lower) system state trajectories comparison among iLQR, Quattro (no auxiliary controller), and Quattro blended (with auxiliary of LQR). The horizon is 30, and the Transformer prediction length is 25.

Table II shows that our accelerator delivers the expected performance in Transformer inference. In the cart-pole control scenario, it achieves a similar speed to the Apple M4 Pro@3.5 GHz, while running **1.55× faster than the NVIDIA RTX4070 and 17.67× faster than the Cortex-A72 in the Raspberry Pi**. In the quadrotor control scenario, our accelerator is **1.8× faster than the M4 Pro CPU and 20.8× faster than the Raspberry Pi**. Although the inference time is close to that of the RTX4070 in the quadrotor task, our accelerator uses much less power, consuming only 1.15 W for the cart-pole and 1.68 W for the quadrotor, compared to the steady 13 W drawn by the GPU in both cases.

The hardware resource overhead is extremely low on a mid-range FPGA, with average utilization remaining below 10% for all resource types except BRAM, which is primarily used to store model parameters. Under the same 20nm process, the estimated silicon area is significantly smaller than that of typical CPUs or GPUs.

E. Trajectory Tracking Performance

Our framework effectively controls both systems to follow desired trajectories. Figure 5 (lower plot) demonstrates that the transformer-assisted (Quattro) controller closely matches

TABLE II: FPGA Accelerator Performance Metrics

Metric	Cart-Pole	Quadrotor
Inference Performance		
Latency (ms)	1.05	1.73
Speed-Up (vs. 10-core M4 Pro CPU)	0.99×	1.8×
Speed-Up (vs. RTX4070 GPU)	1.55×	1.01×
Speed-Up (vs. Quad-core Cortex-A72)	17.67×	**20.8**×
Power Reduction (vs. RTX4070 GPU)	**11.3**×	7.74×
Resource Utilization		
BRAM Usage (%)	24	41
DSP Usage (%)	4	9
FF Usage (%)	4	3
LUT Usage (%)	9	8

the ground truth iLQR performance in stabilizing the cart-pole system. Switching to an LQR controller when the system is near its target helps it converge smoothly to the desired state.

We also tested trajectory tracking with a more complex 8-shaped trajectory for the quadrotor system. As shown in Figure 6, the Quattro-controlled system accurately tracks the reference trajectory across various prediction lengths. The predicted trajectories consistently align closely with the ground truth computed by full-horizon iLQR (50 steps).

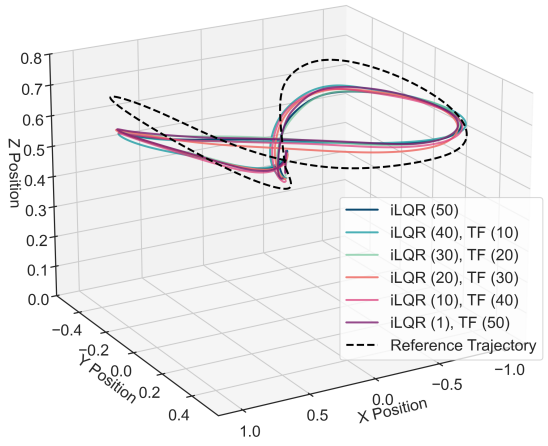

Fig. 6: Quadrotor trajectory tracking performance under varying allocations of iLQR and Transformer computation steps, demonstrating accurate tracking relative to the reference trajectory.

F. Hyperparameter Exploration

The accuracy of the Transformer model predictions depends on the lengths of the input sequences for both states and gains. As shown in Figure 4, performance varies with the number of initial iLQR steps before switching to the Transformer. To balance prediction accuracy and inference time, we chose 5 iLQR steps for the cart-pole system and 1 step for the quadrotor.

We further explored key Transformer hyperparameters, including the number of attention heads (`nhead`) and the model dimension (`dmodel`), to validate our design choices. Figure 7 presents the results. For the cart-pole system, the configuration `dmodel`=128 and `nhead`=4 achieved the lowest MSE (0.0053) with a fast inference time (0.9ms). For the quadrotor, the same configuration yielded the lowest inference time (2.0ms) and a good MSE (0.2051), highlighting a consistent trade-off across tasks.

As shown in Figure 7, smaller models generally perform better in both accuracy and speed, while increasing `dmodel` or `nhead` beyond a certain point leads to diminishing returns or higher latency. The selected configuration balances model capacity with runtime efficiency: `dmodel`=128 provides sufficient expressiveness without excessive cost, and `nhead`=4 allows effective multiple attention. These results confirm the suitability of lightweight architectures for real-time deployment.

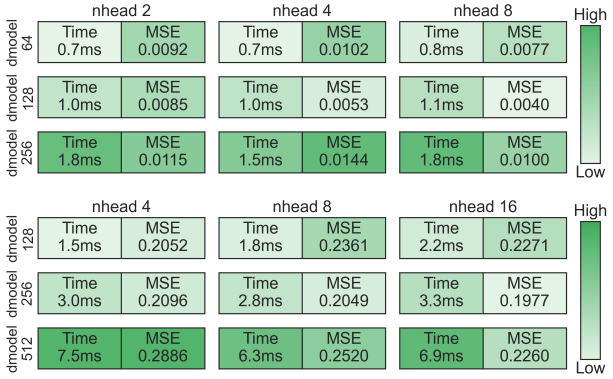

Fig. 7: Hyperparameter exploration of Transformer models on cart-pole (upper) and quadrotor (lower). Each cell shows MSE and inference time for different `dmodel` and `nhead` settings. The lighter green cell indicates better overall performance.

V. CONCLUSION

In this paper, we presented Quattro, a Transformer-accelerated iLQR framework that addresses the sequential bottleneck in traditional iLQR algorithms through parallel inference of control gains. By predicting intermediate computational variables, Quattro maintains the structure of optimal control while significantly reducing computation time. Our experiments on cart-pole and quadrotor systems demonstrate that Quattro achieves up to 27× per-iteration speedup and 17.8× end-to-end acceleration within an MPC framework. FPGA-based deployment further shows 17–20× speedups over edge CPUs and up to 1.55× than GPU with 11.3× power reduction and dramatically low hardware overhead. While larger Transformer models can yield even higher prediction accuracy (sometimes surpassing the original iLQR), they introduce higher latency and hardware overhead. Therefore, we select a lightweight configuration (`dmodel`=128, `nhead`=4) that balances accuracy, efficiency, and deployability. These results confirm the potential of Transformer-based acceleration for real-time optimal control. Future work will extend Quattro to more complex robotic systems, investigate its robustness to model mismatch, conduct a formal stability analysis of the blended controller, and explore techniques such as adaptive prompt tuning for efficient online adaptation.

REFERENCES

[1] G. Bledt, M. J. Powell, B. Katz, J. Di Carlo, P. M. Wensing, and S. Kim, "Mit cheetah 3: Design and control of a robust, dynamic quadruped robot," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 2245–2252.

[2] S. H. Jeon, S. Kim, and D. Kim, "Online optimal landing control of the mit mini cheetah," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 178–184.

[3] G. Romualdi, S. Dafarra, G. L'Erario, I. Sorrentino, S. Traversaro, and D. Pucci, "Online non-linear centroidal mpc for humanoid robot locomotion with step adjustment," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 10 412–10 419.

[4] D. G. Nguyen, S. Park, J. Park, D. Kim, J. S. Eo, and K. Han, "An mpc approximation approach for adaptive cruise control with reduced computational complexity and low memory footprint," *IEEE Transactions on Intelligent Vehicles*, vol. 9, no. 2, pp. 3154–3167, 2023.

[5] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 4906–4913.

[6] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 1168–1175.

[7] M. Neunert, M. Stäuble, M. Giftthaler, C. D. Bellicoso, J. Carius, C. Gehring, M. Hutter, and J. Buchli, "Whole-body nonlinear model predictive control through contacts for quadrupeds," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1458–1465, 2018.

[8] T. A. Howell, B. E. Jackson, and Z. Manchester, "Altro: A fast solver for constrained trajectory optimization," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 7674–7679.

[9] J. Zhu, J. J. Payne, and A. M. Johnson, "Convergent ilqr for safe trajectory planning and control of legged robots," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 8051–8057.

[10] B. Plancher and S. Kuindersma, "A performance analysis of parallel differential dynamic programming on a gpu," in *Algorithmic Foundations of Robotics XIII: Proceedings of the 13th Workshop on the Algorithmic Foundations of Robotics 13*. Springer, 2020, pp. 656–672.

[11] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, "A survey of accelerator architectures for deep neural networks," *Engineering*, vol. 6, no. 3, pp. 264–274, 2020.

[12] T. Cai, Y. Li, Z. Geng, H. Peng, J. D. Lee, D. Chen, and T. Dao, "Medusa: Simple llm inference acceleration framework with multiple decoding heads," *arXiv preprint arXiv:2401.10774*, 2024.

[13] E. Pellegrini and R. P. Russell, "A multiple-shooting differential dynamic programming algorithm. part 1: Theory," *Acta Astronautica*, vol. 170, pp. 686–700, 2020.

[14] M. Giftthaler, M. Neunert, M. Stäuble, J. Buchli, and M. Diehl, "A family of iterative gauss-newton shooting methods for nonlinear optimal control," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1–9.

[15] Y. Lee, M. Cho, and K.-S. Kim, "Gpu-parallelized iterative lqr with input constraints for fast collision avoidance of autonomous vehicles," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 4797–4804.

[16] C. Dai, J. Su, and J. Wang, "A parallel iterative linear quadratic controller for autonomous trajectory optimization," in *2024 43rd Chinese Control Conference (CCC)*. IEEE, 2024, pp. 4573–4578.

[17] H. Li, W. Yu, T. Zhang, and P. M. Wensing, "A unified perspective on multiple shooting in differential dynamic programming," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 9978–9985.

[18] S. H. Hong, J. Ou, and Y. Wang, "Physics-guided neural network and gpu-accelerated nonlinear model predictive control for quadcopter," *Neural Computing and Applications*, vol. 35, no. 1, pp. 393–413, 2023.

[19] D. Celestini, D. Gammelli, T. Guffanti, S. D'Amico, E. Capello, and M. Pavone, "Transformer-based model predictive control: Trajectory optimization via sequence modeling," *IEEE Robotics and Automation Letters*, 2024.

[20] V. Zinage, A. Khalil, and E. Bakolas, "Transformermpc: Accelerating model predictive control via transformers," *arXiv preprint arXiv:2409.09266*, 2024.

[21] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[23] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," *arXiv preprint arXiv:1506.00019*, 2015.

[24] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[25] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, 2019, pp. 4171–4186.

[26] D. Soydaner, "Attention mechanism in neural networks: where it comes and where it goes," *Neural Computing and Applications*, vol. 34, no. 16, pp. 13 371–13 385, 2022.

[27] NVIDIA Corporation. (2025) DLSS 4 Introduces Multi Frame Generation and Transformer-Based AI Models. Accessed: 2025-03-19. [Online]. Available: https://www.nvidia.com/en-us/geforce/news/dlss4-multi-frame-generation-ai-innovations/

[28] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[29] B. Li, S. Pandey, H. Fang, Y. Lyv, J. Li, J. Chen, M. Xie, L. Wan, H. Liu, and C. Ding, "Ftrans: energy-efficient acceleration of transformers using fpga," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2020, pp. 175–180.

[30] Z. Zhao, R. Cao, K.-F. Un, W.-H. Yu, P.-I. Mak, and R. P. Martins, "An fpga-based transformer accelerator using output block stationary dataflow for object recognition applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 70, no. 1, pp. 281–285, 2022.

[31] S. Tunyasuvunakool, A. Muldal, Y. Doron, S. Liu, S. Bohez, J. Merel, T. Erez, T. Lillicrap, N. Heess, and Y. Tassa, "dm_control: Software and tasks for continuous control," *Software Impacts*, vol. 6, p. 100022, 2020.

[32] K. Zakka, Y. Tassa, and MuJoCo Menagerie Contributors, "Mujoco menagerie: A collection of high-quality simulation models for mujoco," http://github.com/google-deepmind/mujoco_menagerie, 2022, accessed: 2025-03-31.

[33] H. Chen, N. Zhang, S. Xiang, Z. Zeng, M. Dai, and Z. Zhang, "Allo: A programming model for composable accelerator design," *Proc. ACM Program. Lang.*, vol. 8, no. PLDI, jun 2024. [Online]. Available: https://doi.org/10.1145/3656401

[34] L. Dong, S. Xu, and B. Xu, "Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition," in *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2018, pp. 5884–5888.

[35] K. A. A. Fuad and L. Chen, "A survey on sparsity exploration in transformer-based accelerators," *Electronics*, vol. 12, no. 10, p. 2299, 2023.

[36] L. Liu, Z. Qu, Z. Chen, F. Tu, Y. Ding, and Y. Xie, "Dynamic sparse attention for scalable transformer acceleration," *IEEE Transactions on Computers*, vol. 71, no. 12, pp. 3165–3178, 2022.

[37] J. Park, H. Yoon, D. Ahn, J. Choi, and J.-J. Kim, "Optimus: Optimized matrix multiplication structure for transformer neural network accelerator," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 363–378, 2020.

[38] X. Yang and T. Su, "Efa-trans: An efficient and flexible acceleration architecture for transformers," *Electronics*, vol. 11, no. 21, p. 3550, 2022.

[39] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[40] M. D. McKay, R. J. Beckman, and W. J. Conover, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 42, no. 1, pp. 55–61, 2000.

[41] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.