POLITECNICO DI TORINO
Repository ISTITUZIONALE

Before Parc Fermé: RL-Time Pruning for Efficient Embodied LLMs in Autonomous Driving

Original
Before Parc Fermé: RL-Time Pruning for Efficient Embodied LLMs in Autonomous Driving / Benfenati, L., Azimi, A., Risso, M., Carapellese, F., Pagliari, D., Burrello, A.. - (In corso di stampa). (2026 Conference on Robot Learning (CoRL) Austin, Texas (USA) November 9-12, 2026).

Availability:
This version is available at: 11583/3016196 since: 2026-10-01T10:22:06Z

Publisher:
Proceedings of Machine Learning Research (PMLR)

Published
DOI:

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

08 November 2026

arXiv:2605.31256v1 [cs.RO] 29 May 2026

Before Parc Fermé: RL-Time Pruning for Efficient Embodied LLMs in Autonomous Driving

Luca Benfenati[†,*], **Ali Azimi**[†,*], **Matteo Risso**[†],
Fabio Carapellese[‡], **Daniele Jahier Pagliari**[†] **Alessio Burrello**[†]

[†]Department of Control and Computer Engineering,
[‡]Department of Mechanical and Aerospace Engineering,
Politecnico di Torino

Abstract: Embodied Large Language Models (LLMs) are increasingly used as reasoning modules in robotic control pipelines to improve human-robot interaction, but their memory and generation latency make real-time deployment difficult. Pruning can reduce these costs, but for controllers that undergo multiple pre- and post-training phases, the crucial question is not only how much to prune, but when pruning should occur.

In this work, we propose *Before Parc Fermé* (BPF), a pruning strategy performed during RL that compresses embodied LLM controllers while they are still being optimized for closed-loop behavior. This allows pruning decisions to account for the task-specific supervision and closed-loop feedback that shape the final controller. We propose two variants: BPF-RL, which performs iterative pruning during RL by removing part of the model at predefined training intervals, and BPF-SFT/RL, which first prunes part of the model structure during SFT and then further compresses it during RL using the same iterative strategy as BPF-RL until the target pruning ratio is reached. We evaluate BPF on RobotxR1, an LLM-based autonomous-driving control pipeline, using an established LLM pruning framework (LLM-Pruner), and compare it against post-training pruning, post-training pruning with RL recovery, SFT-stage pruning, and smaller dense models from the same family.

Our results show that BPF provides the best task-performance vs. memory and throughput trade-off among the considered pruning strategies. When compressing the larger RobotxR1 models, BPF-SFT/RL achieves a $1.69\times$ better size–end-to-end performance trade-off than directly selecting a smaller dense model from the same family, measured as removed parameters per lost percentage point of control adaptability. On the Jetson AGX Orin mounted on the target robotic platform, the compact models improve decode throughput by up to 27%. Finally, we further show that pruning must be evaluated end-to-end, since changes in generated output tokens can affect downstream latency and closed-loop behavior.

Keywords: Large Language Models, Embodied AI, RL-Time Pruning, Autonomous Driving

1 Introduction

Large language models (LLMs) are increasingly used in robotic systems as high-level interfaces between human commands and machine actions [1, 2, 3, 4, 5]. Through large-scale pretraining, these models acquire broad linguistic, task-level, and commonsense knowledge, which makes them useful for interpreting underspecified natural-language commands and adapting them to the current

[*]Denotes equal contribution.

robotic context. However, even relatively small LLMs introduce non-negligible memory, latency, and energy overheads, making them a difficult target for real-time local robotic deployment [6].

Recently, RobotxR1 [7] studied the use of LLMs for autonomous driving control, where the model is used to translate natural language user-specified driving instructions into closed-loop vehicle behavior. RobotxR1 uses two LLM-based modules: DecisionxR1 evaluates whether the current driving behavior matches the user instruction, while MPCxR1 adapts Model Predictive Control (MPC) parameters to realize the desired behavior. Both modules are trained with supervised fine-tuning (SFT) followed by GRPO-based [8] reinforcement learning (RL), allowing their behavior to improve through interaction with the driving environment.

Despite the competitive task performance of RobotxR1 [7], the gap highlighted in the results between larger and smaller models like Qwen2.5-3B and Qwen2.5-1.5B [9] illustrates a central deployment problem: the larger model improves control adaptability by more than 20%, but increases LLM token generation throughput by more than 60%, making real-time deployment harder. This creates a trade-off between the model scale that learns robust embodied reasoning and the one that can be efficiently executed on a constrained robotic platform. Pruning is a natural way to address this issue, since it can reduce the memory and computational cost of larger LLMs while retaining part of their task-performance advantage over smaller manually-designed models.

However, the correct way to apply it in an embodied LLM system trained with SFT and RL is not obvious and has not yet been studied extensively. If pruning is applied after SFT but before RL, the pruning criterion has no access to the closed-loop feedback that determines which structures matter for the final driving task. If pruning is applied only after SFT and RL, compression acts on the final task-adapted model, but the remaining weights have limited opportunity to adapt to the removed structures. Pruning during RL addresses both issues: the model has already learned the supervised task format, and further closed-loop feedback is available before training has fully converged.

Based on this observation, we propose *Before Parc Fermé*[1] (BPF), a family of RL-coupled pruning recipes for embodied LLM controllers. BPF compresses the model while it is still being optimized for closed-loop behavior, rather than treating pruning as a purely post-training operation. We study two variants: *BPF-RL*, which performs iterative pruning during RL, and *BPF-SFT/RL*, which first prunes part of the model structure during SFT and then further compresses the remaining structures during RL using the same iterative strategy as BPF-RL until the target compression ratio is reached. Our central research thesis is that a larger LLM compressed with BPF during adaptation can preserve more reasoning and control ability than models pruned at other points of the SFT-RL pipeline, including before RL and after training.

We evaluate BPF on both RobotxR1 LLM-based modules, DecisionxR1 and MPCxR1.

Our contributions are summarized as follows. **(i)** We introduce BPF, a pruning recipe that couples RL and structured pruning for embodied LLM controllers, using RobotxR1 as a representative SFT/RL-trained robotic LLM system. We show that this strategy is more effective than pruning only after training, pruning before RL, or applying post-RL recovery. Our best BPF-SFT/RL strategy improves task performance by up to 11.4% with respect to doing pruning and post-RL recovery. **(ii)** We characterize the resulting deployment trade-off in terms of task performance, memory footprint, generation throughput, and latency. In particular, BPF-SFT/RL achieves a $1.69\times$ better size–end-to-end performance trade-off than directly selecting the smaller Qwen2.5-1.5B model, measured as removed parameters per lost percentage point of control adaptability, while improving decode throughput by 27% on the Jetson AGX Orin mounted on the target robotic platform. **(iii)** We show that pruning embodied LLM controllers must be evaluated end-to-end, not only through task-agnostic metrics (e.g., memory footprint, generation throughput). Since LLM latency depends on the generated-token count, which can change after pruning, compression can alter downstream latency and behavior. This is especially relevant in RobotxR1, where DecisionxR1 outputs condition MPCxR1 generation.

[1] *Parc fermé* refers to the Formula 1 period in which cars are locked from further modification before the race. Here, it denotes the post-training stage where the model architecture is fixed.

2 Related Work

LLMs for Robot Control and Autonomous Driving: Recent work has studied LLMs as high-level reasoning modules for robotic systems, where the language model interprets instructions, decomposes tasks, selects skills, or generates objectives for a downstream controller [1, 2, 3, 4, 5]. In autonomous driving, LLMs and multimodal LLMs have been used for scene interpretation, decision-making, and planning [10, 11, 12]. Baumann et al. [13] introduces the on-board LLM-based autonomous-driving pipeline used in this work, where a decision model interprets driving instructions, and an MPC-oriented language model adapts controller parameters. Boyle et al. [7] extend this pipeline by adapting the *R1-Zero* training protocol [8] to the same closed-loop autonomous-driving environment, bringing reasoning-oriented LLMs into the robotic control setting and showing that they outperform their non-reasoning counterparts. Building on this setup, our work studies whether pruning during learning is a better alternative to replacing the model with a smaller dense one from the same family, and investigates the optimal training stage for pruning.

Post-Training LLM Pruning: Recent LLM-specific pruning methods reduce model size and inference cost by applying unstructured or structured sparsification to pretrained models [14, 15, 16, 17, 18, 19]. In particular, RL-Pruner [19] further uses RL to recover from pruning-induced loss in encoder-based language models without retraining. While all these methods provide practical mechanisms for compressing LLMs, they treat pruning as a compression step applied to an already-trained model, rather than as part of task adaptation or reasoning-oriented training.

Learning-Aware LLM Pruning: A more relevant direction for our work is pruning that is integrated with training or fine-tuning. Sheared LLaMA [20] uses targeted structured pruning to reduce a larger pretrained LLM to a specified smaller architecture, followed by continued pretraining, showing that starting from a larger model can be more effective than training a smaller model directly. Pruning-Aware Tuning [21] incorporates structured pruning into finetuning by learning pruning decisions jointly with task adaptation, rather than fixing the sparsity pattern before finetuning. Adapt-Pruner [22] further studies adaptive structured pruning during training for efficient small language model construction. Closest in spirit to our setting, reinforcement learning has also been used to learn structured pruning distributions for CNN compression [23], however, this work optimizes a pruning policy for CNNs, rather than pruning an LLM during RL-based embodied control training.

Our work studies learning-aware pruning in embodied robotics [7] and in an *R1-Zero*-style setup [8], where the model is adapted through SFT and closed-loop RL to acquire reasoning behavior. We study pruning as a training-time design choice, comparing compression after training, during SFT, during RL, and across both stages. This distinction matters because the relevant costs in robotics are not only model size but also inference latency, generated token length, and downstream effects across the control pipeline.

3 Methodology

This work builds on RobotxR1 [7], which adapts LLMs for robotic reasoning through SFT followed by closed-loop RL. We keep the controller, prompts, rewards, and evaluation protocol unchanged, and study how structured pruning can be integrated into the same training pipeline. The key methodological question is where to apply pruning so that compression can exploit closed-loop training feedback without being delayed until adaptation has fully converged.

3.1 RobotxR1 Pipeline

We use the same DecisionxR1-MPCxR1 pipeline as RobotxR1 [7]. DecisionxR1 is a RAG-enhanced LLM that receives the user command and recent robot state history and determines whether the current vehicle behavior matches the requested behavior. The state history contains track-relative information such as vehicle position with respect to the racing line, heading error, steering angle, and

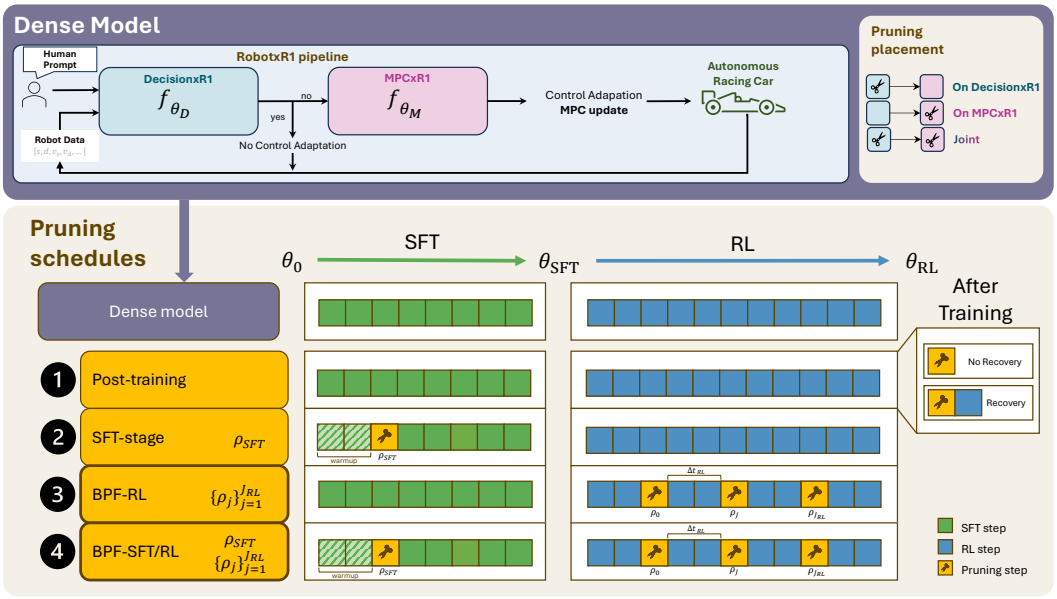


Figure 1: Overview of the pruning strategies and placement in our pipeline.

longitudinal velocity. MPCxR1 then uses the DecisionxR1 output, the user command, the robot state history, retrieved context, and the MPC formulation to generate textual updates to the exposed MPC parameters. The parsed parameters are applied to the low-level MPC controller, which executes the corresponding behavior in closed loop.

Both modules follow the same high-level adaptation sequence with SFT followed by RL. During SFT, the models are trained with a next-token prediction loss on task-specific instruction-response data. During RL, DecisionxR1 is optimized with behavior-adherence and formatting rewards, while MPCxR1 is optimized through closed-loop simulation using driving-performance, formatting, and parameter-validity rewards. This shared SFT-RL structure defines the training pipeline in which we introduce pruning. Additional details on the inherited RobotxR1 module inputs, outputs, and rewards are reported in Appendix A.

Because DecisionxR1 and MPCxR1 are executed sequentially, pruning can have both local and downstream effects. Pruning DecisionxR1 can change the text received by MPCxR1, while pruning MPCxR1 directly affects controller-parameter generation. In the experiments, we therefore apply the same pruning schedules to DecisionxR1, MPCxR1, and both modules jointly.

3.2 BPF Pruning Schedule

BPF defines when pruning is introduced within the SFT-RL training pipeline. Since DecisionxR1 and MPCxR1 share this training structure, we define all schedules for a generic LLM module with weights θ and apply them to either module in the experiments.

Following Fig. 1, we compare pruning after the full SFT-RL pipeline, during SFT, during RL, and across both stages. Our main focus is on the two RL-coupled schedules: incremental pruning during RL (BPF-RL), and split SFT-RL pruning (BPF-SFT/RL), where part of the model is pruned during SFT, and the remaining compression follows the same incremental RL schedule as BPF-RL.

Post-training pruning. The simplest pruning schedule is **post-training** pruning (PTP), denoted as ① in Fig. 1, where the model is first adapted with the standard SFT and RL, obtaining θ_{RL}, and is then pruned in one-shot. This corresponds to the conventional setting in which compression is applied only after task adaptation has completed. We consider both the direct post-training variant

(PTP) and a recovery variant (PTP+R), where the pruned model is further optimized with additional RL steps to recover the lost performance.

SFT-stage pruning. Fig. 1-② depicts **SFT-stage** pruning schedule, where pruning is applied as a single compression step during SFT, before the RL stage starts. Starting from θ_0, the model is first trained for an SFT warmup phase without pruning. After this warmup, the model is pruned with target ratio ρ_{SFT}, and the remaining SFT steps continue on the compressed model, yielding $\theta^{\rho}_{\mathrm{SFT}}$. RL is then run on this already compressed model.

BPF-RL. Fig. 1-③ depicts our first RL-coupled pruning schedule, **BPF-RL**. The model is first trained with SFT without pruning, obtaining θ_{SFT}. Compression is then introduced during RL, so that pruning decisions are made after the model has learned the supervised task format, but while it is still being optimized with closed-loop feedback. Rather than pruning once, BPF-RL removes weights iteratively during RL. Let ρ be the final target pruning ratio and let J_{RL} be the number of pruning events. We split the target compression uniformly across pruning events, so that each event increases the cumulative pruning ratio by ρ/J_{RL}. Starting from θ_{SFT}, RL is run for an initial warm-up interval. Then, every Δt_{RL} RL steps, pruning scores are recomputed on the current model, and the lowest-scoring parameter groups are removed until the cumulative pruning ratio reaches $j\rho/J_{\mathrm{RL}}$ at pruning event j. After each pruning event, RL resumes under the same reward objective. This alternation between RL updates and pruning allows the remaining weights to adapt to the removed structures while the model is still being optimized for the driving task.

BPF-SFT/RL. Fig. 1-④ shows our second RL-coupled pruning schedule, **BPF-SFT/RL**. This schedule is motivated by the fact that SFT and RL provide complementary information. SFT exposes the model to task-specific prompts, output formats, and supervised input-output behavior, making it a useful stage to remove clearly redundant structure. However, SFT does not provide closed-loop feedback from the final driving task. For this reason, BPF-SFT/RL applies only a partial compression during SFT and leaves the remaining compression to RL, where pruning scores can reflect the model after it has started adapting to closed-loop behavior.

Concretely, let ρ be the final target pruning ratio. BPF-SFT/RL first prunes the model during SFT up to an intermediate ratio $\rho_{\mathrm{SFT}} < \rho$, after an SFT warm-up phase. The remaining compression, $\rho - \rho_{\mathrm{SFT}}$, is then reached during RL using the same iterative procedure as BPF-RL. If J_{RL} pruning events are used during RL, each event adds $(\rho - \rho_{\mathrm{SFT}})/J_{\mathrm{RL}}$ to the cumulative pruning ratio. Thus, BPF-SFT/RL combines partial compression after supervised task adaptation with RL-time compression, allowing the model to access task feedback that determines the final control behavior.

3.3 Taylor-Based Pruning

The pruning schedules described in previous sections are agnostic to the specific pruning criterion and can be instantiated with either structured or unstructured pruning. In this work, to demonstrate BPF performance, we adopt LLM-Pruner [16], an LLM pruning framework that can operate at different granularities. In the structured setting, the method first identifies groups of dependent parameters that must be removed jointly to preserve architectural consistency after compression. In the unstructured setting, the same formulation is recovered by treating each scalar weight as an independent group.

Given a calibration batch of data $\mathcal{D}$ and a loss $\mathcal{L}(\mathcal{D};\theta)$ evaluated at the current parameters θ, the importance of a scalar parameter $w_i \in \theta$ is approximated with a first-order Taylor score:

$$I(w_i) = \left| w_i \frac{\partial \mathcal{L}(\mathcal{D};\theta)}{\partial w_i} \right| \tag{1}$$

For a coupled group $\mathcal{G}_k$, we aggregate the scores of its parameters as $S(\mathcal{G}_k) = \sum_{w_i \in \mathcal{G}_k} I(w_i)$. Groups with the lowest scores are removed until the target pruning ratio for that pruning point is reached.

In this work, we focus on structured pruning because it can more straight-forwardly yield practical inference speedups [24, 18]. This better aligns with common deployment settings for standard

inference runtimes and hardware, whereas unstructured sparsity often requires dedicated kernels and hardware to achieve practical speedups. We also report unstructured pruning results in the Appendix C.2.

4 Experiments and Results

All experiments are conducted on the RobotxR1 pipeline [7], using the same controller, prompts, rewards, and closed-loop evaluation protocol described in Sec. 3.1. Additional setup details are provided in Appendix B.

4.1 DecisionxR1 Pruning with Fixed MPCxR1

We first isolate the effect of pruning DecisionxR1, while leaving MPCxR1 unpruned. Following the control-adaptability metric used in RobotxR1, we report the average percentage improvement over the default MPC controller across four task errors, (E_c, E_v, E_r, E_s). Here, the default MPC is the original hand-tuned racing-line controller before any LLM-generated parameter update. Additional details are provided in Appendix A.

Fig. 2 shows the resulting trade-off between the DecisionxR1 model size and end-to-end control adaptability. The x-axis reports the number of parameters, while the y-axis reports the average improvement over the default MPC controller. Square markers denote the unpruned baselines. Starting from Qwen2.5-3B, we test 10%, 20%, 30%, and 45% pruning ratios for PTP (orange), PTP+R (red), BPF-RL (purple), and BPF-SFT/RL (blue). SFT-stage pruning is omitted because it fails the task even at the smallest pruning ratio. Additional results obtained with best BPF-SFT/RL recipe starting from 1.5B baseline are reported in Appendix C.1.

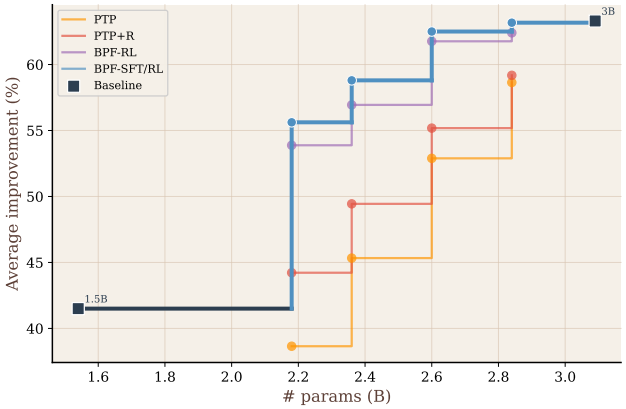


Figure 2: Average control-adaptability improvement over the default MPC controller as in [7] versus model size.

The results show that our novel BPF-RL and -SFT/RL give the best trade-off among the schedules applied to the 3B DecisionxR1 model. In particular, for the smallest architecture, obtained at 45% pruning ratio, BPF-RL improves the average control-adaptability score by 9.7%/15.2% over PTP+R/PTP, while BPF-SFT/RL improves it by 11.4%/17%. The same point also provides the best trade-off between the dense 3B and 1.5B models, achieving 0.91B parameter reduction with 7.61% control-adaptability loss relative to the dense 3B model, corresponding to a ratio of 0.12B parameter reduction per lost percentage point. By contrast, moving directly to the dense 1.5B backbone gives 1.55B parameter reduction losing 21.7% points, yielding only 0.071B parameter reduction per lost point. Thus, BPF-SFT/RL provides a $1.69\times$ better compression–adaptability trade-off than selecting the smaller baseline model. Moreover, the BPF-SFT/RL model with a 10% pruning ratio achieves the same level of control adaptability as the strongest RobotxR1 baseline while reducing its parameter count by 0.25B.

4.2 Joint DecisionxR1-MPCxR1 Pruning

We next evaluate full-pipeline pruning, where both DecisionxR1 and MPCxR1 are pruned. Since DecisionxR1 output is passed to MPCxR1, this setting captures how compression propagates through the full RobotxR1 control pipeline.

Fig. 3 reports on the x-axis the total number of parameters across the two LLM modules, while the y-axis reports the average improvement over the default MPC controller. Different colors indicate the DecisionxR1 pruning ratio, while different marker shapes indicate the MPCxR1 pruning ratio with Pareto labels reporting DecisionxR1/MPCxR1 pruning ratios. The labeled Pareto points show that moderate MPCxR1 pruning is critical. The $30/10$ configuration keeps the pipeline close to the unpruned 3B baseline, with less than 5% loss in average improvement, whereas the $30/30$ configuration drops to roughly the mid-40% range. Thus, increasing MPCxR1 pruning from 10% to 30% causes a substantially larger degradation than increasing DecisionxR1 pruning alone. This confirms that MPCxR1 is the more sensitive module, consistent with its role in directly generating the MPC-parameter update used by the low-level controller.

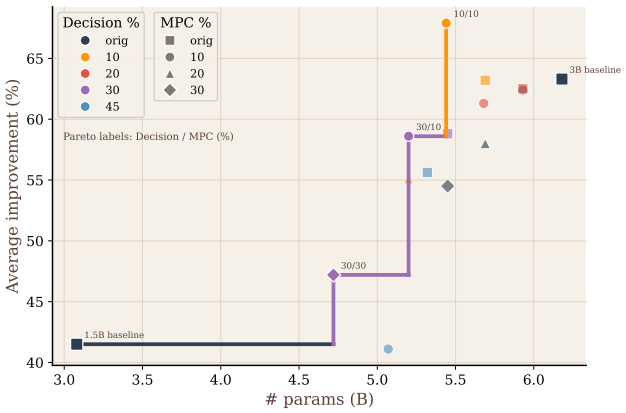


Figure 3: End-to-end control-adaptability trade-off when pruning both DecisionxR1 (colors) and MPCxR1 (shapes).

The same trend holds when both modules are pruned jointly. Light compression of the full cascade can even improve over the unpruned 3B baseline. Indeed, the $10/10$ configuration achieves the best Pareto point, suggesting that BPF removes parameters that are not beneficial for the final closed-loop task. However, the gain is module-dependent. The best trade-off configurations keep MPCxR1 only mildly pruned, 10%, while allowing larger compression in DecisionxR1, up to 30–45%.

4.3 Embedded Deployment

To assess whether our BPF models provide practical deployment benefits, we deploy all pruned configurations identified in Sec . 4.2 on the physical robotic platform shown in Fig. 4. The vehicle is equipped with a Jetson AGX Orin, which acts as the onboard inference device for the LLM modules. Additional measurements reported in Appendix C.4.

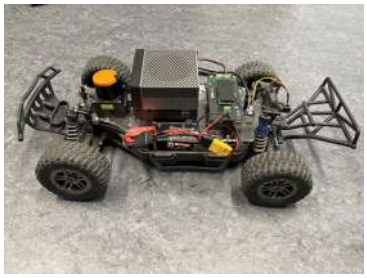


Figure 4: Robotic platform equipped with the Jetson AGX Orin used for deployment.

The deployment results are summarized in Table 1. For fairness, all dense baselines are redeployed and remeasured on our deployment stack, where they achieve higher throughput than in the original RobotxR1 implementation [7]. All pruned models are obtained with the BPF-SFT/RL recipe. After pruning, each model is quantized to `Q5_K_M`, exported to `GGUF` format, and deployed with the `llama.cpp` runtime [25]. Before deployment, we apply a compilation pass that zero-pads MLP weights so that output dimensions are multiples of 256, avoiding FP16 fallbacks in `llama.cpp` for non-aligned layers. Memory footprint and decode throughput are measured with the `llama-bench` utility.

Table 1 shows that BPF consistently improves inference throughput while reducing memory footprint. On the Jetson AGX Orin, pruning DecisionxR1 by 45% reduces memory from 2.07GB to 1.51GB and increases decode throughput from 45 to 58 tok/s. A similar trend is observed for MPCxR1 pruning: at 30% ratio, memory decreases to 1.62GB and throughput reaches 55 tok/s.

While Table 1 reports isolated-model throughput, the robot runs DecisionxR1 and MPCxR1 sequentially. We evaluate the complete cascade latency for the BPF-SFT/RL configurations from Sec. 4.2, measuring the number of tokens generated by each stage. Fig. 5 shows that reducing parameters alone does not necessarily reduce end-to-end latency. This effect is orthogonal to prun-

ing itself and could be mitigated by training or prompting the modules to produce shorter outputs, for example, by asking MPCxR1 to generate only the parameters that need to be changed.

Table 1: Embedded deployment metrics on the Jetson AGX Orin.

Module	Configuration	Mem. (GB)	Dec. tok/s
	Baseline 3B	2.07	45
	Baseline 1.5B	1.04	73
Decision	BPF-10%	1.92	49
	BPF-20%	1.77	51
	BPF-30%	1.62	56
	BPF-45%	1.51	58
MPC	BPF-10%	1.92	48
	BPF-20%	1.77	51
	BPF-30%	1.62	55

In this pipeline, pruning DecisionxR1 can change the text passed to MPCxR1, while pruning MPCxR1 can make parameter generation more verbose. In both cases, longer outputs can reduce or completely hide the computational savings of a smaller and higher-throughput model. The best latency-wise BPF-SFT/RL configuration prunes DecisionxR1 by 20% and keeps MPCxR1 unpruned, reducing latency by 19.5% with only a 0.8% drop in task-performance. Compared with the 1.5B baseline, this configuration is 1.57× slower but achieves a 21.0% higher task-performance.

5 Conclusions

We introduced *Before Parc Fermé* (BPF), a strategy that integrates pruning into the training process of embodied LLM controllers, with a particular focus on RL-time compression. Across the considered robotic LLM pipeline, BPF-SFT/RL offers the best trade-off among the evaluated pruning schedules, demonstrating that the timing of compression matters for preserving closed-loop behavior. Our deployment results show that structured pruning can reduce the memory footprint and decode throughput on the target robotic platform. However, smaller modules are not necessarily faster end-to-end: pruning can change generated-token length and make the cascaded pipeline slower despite reducing parameter count. This highlights that compression for modular robotic LLM systems must be evaluated at the full-pipeline level, jointly considering task performance, model size, generated output length, throughput, and latency.

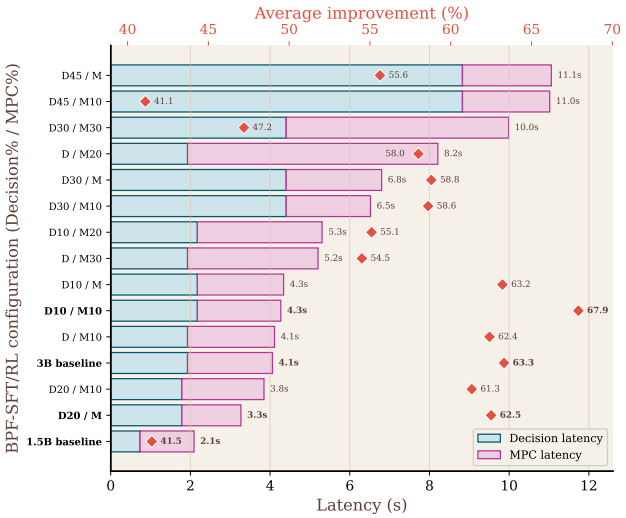


Figure 5: End-to-end latency of the full pipeline. The bottom x-axis reports measured latency, while the top x-axis reports corresponding task-performance from Fig. 3. Pareto configurations are highlighted in **bold**.

Limitations

This work presents four main limitations. First, the two LLM modules are adapted and pruned separately: MPCxR1 is trained on outputs from the original pipeline, not from pruned DecisionxR1 models. This may reduce robustness to distribution shifts introduced by upstream pruning and may contribute to the increased verbosity and latency observed for some pruned cascades, motivating future work on joint or alternating training of compressed modules. Second, we evaluate a limited set of pruning schedules, with fixed pruning intervals and fixed pruning increments; more frequent, smaller, or adaptive RL-time pruning policies remain unexplored. Third, our structured pruning procedure is not quantization-aware, requiring padding to align pruned MLP dimensions with runtime constraints, which could be avoided by jointly optimizing pruning and quantization. Finally, the RL objective does not penalize the length of generated tokens, although our deployment results show that pruning can increase verbosity and end-to-end latency.

Acknowledgments

This publication is part of the project PNRR-NGEU which has received funding from the MUR – DM 117/2023.

References

[1] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. M. J. Ruano, K. Jeffrey, S. Jesmonth, N. J. Joshi, R. C. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. M. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, and M. Yan. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on Robot Learning*, 2022. URL https://api.semanticscholar.org/CorpusID:247939706.

[2] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. R. Florence, and A. Zeng. Code as policies: Language model programs for embodied control. *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500, 2022. URL https://api.semanticscholar.org/CorpusID:252355542.

[3] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, P. Sermanet, T. Jackson, N. Brown, L. Luu, S. Levine, K. Hausman, and brian ichter. Inner monologue: Embodied reasoning through planning with language models. In *6th Annual Conference on Robot Learning*, 2022. URL https://openreview.net/forum?id=3R3Pz5i0tye.

[4] C. H. Song, J. Wu, C. Washington, B. M. Sadler, W.-L. Chao, and Y. Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models. *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2986–2997, 2022. URL https://api.semanticscholar.org/CorpusID:254408960.

[5] S. Ismail, A. Arbues, R. Cotterell, R. Zurbrügg, and C. A. Alonso. Narrate: Versatile language architecture for optimal control in robotics. *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9628–9635, 2024. URL https://api.semanticscholar.org/CorpusID:268513494.

[6] Z. Yu, B. Wang, P. Zeng, H. Zhang, J. Zhang, Z. Wang, L. Gao, J. Song, N. Sebe, and H. T. Shen. A survey on efficient vision-language-action models. *arXiv preprint arXiv:2510.24795*, 2025.

[7] L. Boyle, N. Baumann, P. Sivasothilingam, M. Magno, and L. Benini. Robotxr1: Enabling embodied robotic intelligence on large language models through closed-loop reinforcement learning. In *9th Annual Conference on Robot Learning*, 2025. URL https://openreview.net/forum?id=Ggu7Hh2xnn.

[8] D. Guo, D. Yang, H. Zhang, J. Song, P. Wang, Q. Zhu, R. Xu, R. Zhang, S. Ma, X. Bi, X. Zhang, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025. ISSN 1476-4687. doi:10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

[9] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

[10] Z. Xu, Y. Zhang, E. Xie, Z. Zhao, Y. Guo, K.-Y. K. Wong, Z. Li, and H. Zhao. Drivegpt4: Interpretable end-to-end autonomous driving via large language model. *IEEE Robotics and Automation Letters*, 9:8186–8193, 2023. URL https://api.semanticscholar.org/CorpusID:263605524.

[11] C. Sima, K. Renz, K. Chitta, L. Chen, H. Zhang, C. Xie, P. Luo, A. Geiger, and H. Li. Drivelm: Driving with graph visual question answering. In *European Conference on Computer Vision*, 2023. URL https://api.semanticscholar.org/CorpusID:266435584.

[12] C. Cui, Y. Ma, X. Cao, W. Ye, and Z. Wang. Drive as you speak: Enabling human-like interaction with large language models in autonomous vehicles. *2024 IEEE/CVF Winter Conference on Applications of Computer Vision Workshops (WACVW)*, pages 902–909, 2023. URL https://api.semanticscholar.org/CorpusID:262054629.

[13] N. Baumann, C. Hu, P. Sivasothilingam, H. Qin, L. Xie, M. Magno, and L. Benini. Enhancing autonomous driving systems with on-board deployed large language models, 2025. URL https://arxiv.org/abs/2504.11514.

[14] E. Frantar and D. Alistarh. SparseGPT: Massive language models can be accurately pruned in one-shot. *arXiv preprint arXiv:2301.00774*, 2023.

[15] M. Sun, Z. Liu, A. Bair, and J. Z. Kolter. A simple and effective pruning approach for large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=PxoFut3dWW.

[16] X. Ma, G. Fang, and X. Wang. LLM-pruner: On the structural pruning of large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=J8Ajf9WfXP.

[17] Y. An, X. Zhao, T. Yu, M. Tang, and J. Wang. Fluctuation-based adaptive structured pruning for large language models. In *AAAI Conference on Artificial Intelligence*, 2023. URL https://api.semanticscholar.org/CorpusID:266362404.

[18] S. Ashkboos, M. L. Croci, M. G. do Nascimento, T. Hoefler, and J. Hensman. SliceGPT: Compress large language models by deleting rows and columns. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=vXxardq6db.

[19] G. Yao, Y. Wang, W. Zhang, X. Deng, C. Du, R. Han, Z. Qin, Y. Li, B. Xie, B. Dong, H. Peng, S. Zhu, L. Zhang, Z. Wang, and Z. Zhang. Rl-pruner: Retraining-free global exploration pruning method based on reinforcement learning. In *2025 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2025. doi:10.1109/IJCNN64981.2025.11228409.

[20] M. Xia, T. Gao, Z. Zeng, and D. Chen. Sheared LLaMA: Accelerating language model pretraining via structured pruning. In *Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@NeurIPS 2023)*, 2023. URL https://openreview.net/forum?id=6s77hjBNfS.

[21] Y. Liu, H. Yang, Y. Chen, R. Zhang, M. Wang, Y. Du, and L. Du. Pat: Pruning-aware tuning for large language models. *CoRR*, abs/2408.14721, 2024. URL https://doi.org/10.48550/arXiv.2408.14721.

[22] B. Wang, R. Pan, S. Diao, X. Pan, J. Zhang, R. Pi, and T. Zhang. Adapt-pruner: Adaptive structural pruning for efficient small language model training. *CoRR*, abs/2502.03460, February 2025. URL https://doi.org/10.48550/arXiv.2502.03460.

[23] B. Wang and V. Kindratenko. Rl-pruner: Structured pruning using reinforcement learning for cnn compression and acceleration, 2024. URL https://arxiv.org/abs/2411.06463.

[24] Y. Wang, M. Ma, Z. Wang, J. Chen, S. Liping, Q. Yang, D. Xu, M. Liu, and B. Qin. CFSP: An efficient structured pruning framework for LLMs with coarse-to-fine activation information. In O. Rambow, L. Wanner, M. Apidianaki, H. Al-Khalifa, B. D. Eugenio, and S. Schockaert, editors, *Proceedings of the 31st International Conference on Computational Linguistics*, pages 9311–9328, Abu Dhabi, UAE, Jan. 2025. Association for Computational Linguistics. URL https://aclanthology.org/2025.coling-main.626/.

[25] G. Gerganov and O.-S. Contributors. Llama.cpp. https://github.com/ggerganov/llama.cpp, 2023. Accessed: May 2026.

[26] N. Baumann, E. Ghignone, J. Kühne, N. Bastuck, J. Becker, N. Imholz, T. Kränzlin, T. Y. Lim, M. Lötscher, L. Schwarzenbach, et al. ForzaETH Race Stack—Scaled Autonomous Head-to-Head Racing on Fully Commercial Off-the-Shelf Hardware. *Journal of Field Robotics*, 2024.

Appendix

A RobotxR1 Pipeline and Evaluation Details

This work uses the RobotxR1 pipeline [7] without modifying the controller, prompts, rewards, or evaluation protocol. RobotxR1 builds on a 1:10 scaled racecar autonomy stack [26], where a kinematic MPC tracks a minimum-curvature racing line in curvilinear coordinates. The robot state contains the longitudinal and lateral displacement with respect to the racing line, heading error, steering angle, and longitudinal velocity. The MPC exposes cost weights and constraints, such as velocity bounds and track-boundary inflation, which are the quantities updated by the LLM-generated parameters.

The pipeline consists of two RAG-enhanced LLM modules executed sequentially. DecisionxR1 receives the user command c, the recent robot state history h_t, and auxiliary context z_D, including retrieved memories and formatting instructions:

$$y_D = f_{\theta_D}(c, h_t, z_D), \tag{2}$$

where θ_D denotes the DecisionxR1 weights. The module determines whether the current vehicle behavior matches the requested behavior. Conditioned on this output, MPCxR1 generates a textual update to the MPC parameters:

$$y_M = f_{\theta_M}(c, h_t, y_D, z_M), \tag{3}$$

where z_M includes the MPC formulation, retrieved memories, and output constraints. The parsed parameters from y_M are applied to the low-level MPC controller and evaluated in closed-loop simulation.

Following RobotxR1, control adaptability is measured as the percentage improvement over the default MPC controller across four behavior-specific task errors: centerline tracking error E_c, velocity-tracking error E_v, reversing error E_r, and smooth-driving error E_s. The centerline error E_c measures deviation from the centerline over a lap; E_v measures deviation from the target velocity specified in the prompt; E_r measures deviation from the reverse-driving target velocity; and E_s measures deviation from zero acceleration, used as a proxy for smooth driving. For each error E, the improvement is computed relative to the default MPC error, and the final score is the average over the four errors:

$$\text{AvgImprove} = \frac{100}{4} \sum_{E \in \{E_c, E_v, E_r, E_s\}} \frac{E^{\text{MPC}} - E^{\text{LLM}}}{E^{\text{MPC}}}. \tag{4}$$

Here, E^{MPC} denotes the error obtained with the default MPC parameters, while E^{LLM} denotes the error after applying the LLM-generated MPC update. The closed-loop evaluation uses the same randomized evaluation prompts, simulation setup, and Grand Tour race-track layout as RobotxR1. The evaluation prompts are unseen during SFT and RL training and include perturbed natural-language variants for each behavior category.

Both modules are adapted through SFT followed by GRPO-based RL. During SFT, the models are trained with the standard next-token prediction loss on task-specific instruction-response data. During RL, DecisionxR1 uses behavior-adherence and formatting rewards, while MPCxR1 combines closed-loop driving-performance, formatting, and parameter-validity rewards.

B Experimental Setup

Both LLM modules, DecisionxR1 and MPCxR1, are instantiated from Qwen2.5 [9]. The dense baselines are Qwen2.5-3B and Qwen2.5-1.5B models trained with the standard RobotxR1 SFT–GRPO pipeline; all pruning experiments start from the Qwen2.5-3B checkpoint. SFT is run for 200 steps, with pruning, if applied, after 100 steps. RL is run for 900 steps. When pruning is applied during RL, we use a 400-step warm-up and split the target pruning ratio ρ across three pruning events separated by $\Delta t_{\text{RL}} = 100$ steps, following Sec. 3.2. For BPF-SFT/RL, the target pruning

ratio ρ is split evenly across the two stages: $\rho/2$ is applied during SFT, and the remaining $\rho/2$ is applied iteratively during RL. All schedules use the same total number of SFT and RL steps.

C Additional Results

C.1 Additional DecisionxR1 Pruning Results

Fig. 6 extends the DecisionxR1 pruning analysis of Fig. 2 by also applying BPF-SFT/RL to the 1.5B checkpoint. The results show that the smaller dense model has less pruning headroom. Starting from 1.5B, a mild 10% pruning ratio only slightly reduces average control adaptability, from 41.5% to 40.01%. However, at 20% pruning the score drops to 31%, showing that the performance degradation appears much earlier than when pruning the 3B model.

Fig. 7 reports the same 3B-initialized pruning configurations in terms of average task-error improvement over the 1.5B baseline. Only models initialized from 3B are included. The plot confirms that BPF-based pruning remains consistently above the smaller dense baseline across pruning ratios. At the 10% pruning ratio, BPF-RL slightly outperforms BPF-SFT/RL on this error-improvement metric. However, this local advantage does not translate into a better average control-adaptability score in Fig. 2, where BPF-SFT/RL provides the strongest overall trade-off.

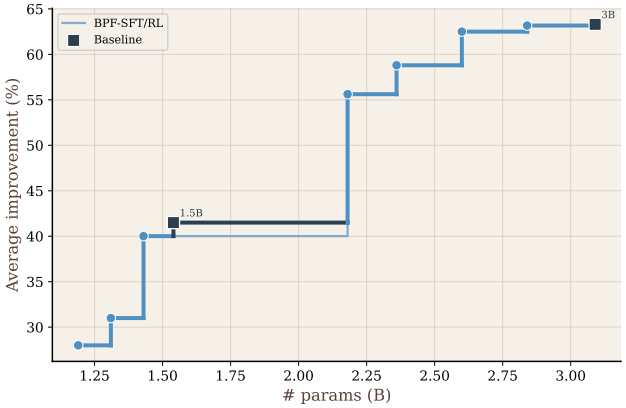


Figure 6: DecisionxR1 pruning with fixed MPCxR1, using both 3B and 1.5B as starting checkpoints for BPF-SFT/RL.

C.2 Unstructured Pruning on DecisionxR1

In the main paper, we focus on structured pruning because it yields smaller dense architectures and is therefore better aligned with standard inference runtimes. For completeness, we also evaluate unstructured pruning on DecisionxR1. This analysis is limited to standalone DecisionxR1 accuracy and does not include end-to-end closed-loop evaluation with MPCxR1. Therefore, these results should be interpreted as a module-level robustness check rather than as evidence of full-pipeline control performance.

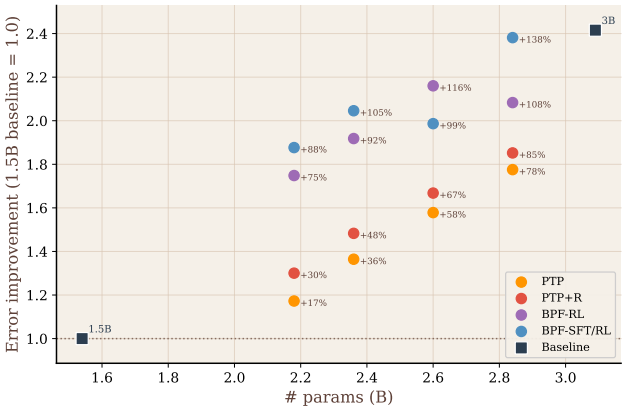
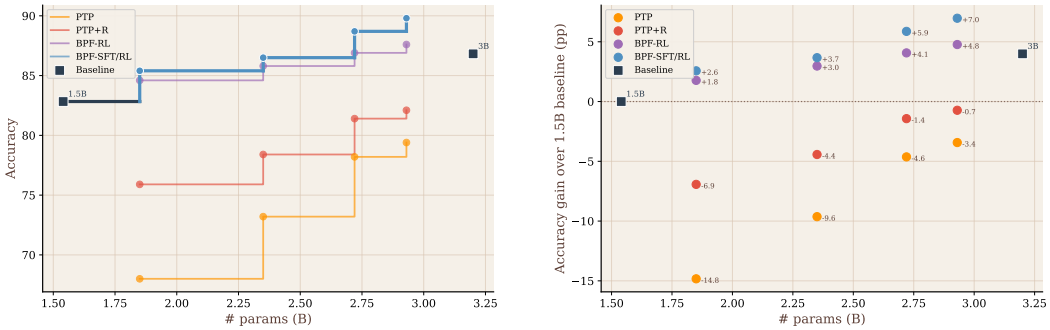


Figure 7: Average task-error improvement of pruned DecisionxR1 models initialized from 3B, measured relative to the 1.5B baseline. The score averages the relative improvement across the four task errors (E_c, E_v, E_r, E_s).

Fig. 8a shows that unstructured pruning preserves standalone DecisionxR1 accuracy over a wide range of parameter budgets. In particular, the pruned 3B-derived BPF models remain above the dense 1.5B baseline for moderate and high pruning ratios. BPF-SFT/RL reaches 89.80% standalone accuracy with 2.93B parameters, slightly outperforming the unpruned 3B DecisionxR1 baseline on this module-level metric. Even at stronger



(a) DecisionxR1 accuracy under unstructured pruning. All pruned models start from 3B, while 1.5B is shown as the smaller dense baseline.

(b) Relative accuracy improvement of unstructured pruned DecisionxR1 models over the dense 1.5B baseline.

Figure 8: Standalone DecisionxR1 results under unstructured pruning.

compression, with 1.85B parameters, BPF-SFT/RL reaches 85.40% accuracy, remaining above the 82.83% accuracy of the dense 1.5B baseline.

Fig. 8b reports the same results as relative accuracy improvement over the dense 1.5B baseline. The improvement remains positive across the reported unstructured BPF configurations, indicating that the larger 3B backbone retains useful redundancy even after a substantial fraction of weights is removed. Interestingly, the post-training pruning variants, PTP and PTP+R, fall below the dense 1.5B baseline, showing that post-training pruning can degrade standalone DecisionxR1 accuracy below the smaller dense baseline when compression is not integrated into adaptation. However, unlike structured pruning, unstructured sparsity does not directly produce a smaller dense architecture and may require sparse kernels or hardware support to translate parameter sparsity into practical inference speedups. For this reason, we use structured pruning for the main end-to-end and deployment experiments, while reporting unstructured pruning here as a module-level robustness check.

C.3 Qualitative Closed-Loop Trajectories under DecisionxR1 Pruning

To complement the quantitative results, Fig. 9 and Fig. 10 show qualitative closed-loop trajectories in the ROS simulator for the prompt *"Stay directly on the middle of the track"*. We compare the original unpruned DecisionxR1-MPCxR1 pipeline (Fig. 9) against four configurations where MPCxR1 is kept unpruned and DecisionxR1 is pruned with BPF-SFT/RL to 10%, 20%, 30%, and 45%, respectively (Fig. 10). In each case, the figure overlays the reference path and the trajectory followed by the vehicle.

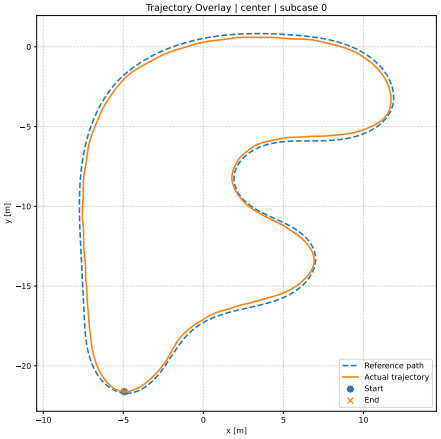


Figure 9: Original unpruned DecisionxR1-MPCxR1 trajectory for the centerline-following prompt.

The results show that the trajectories produced by the pruned DecisionxR1 models remain visually close to the unpruned trajectory in Fig. 9. Even at 45% pruning, the vehicle still follows the centerline behavior with a trajectory comparable to the unpruned baseline. These qualitative results are consistent with the end-to-end control-adaptability scores reported in Sec. 4.1, indicating that the proposed pruning strategy preserves the relevant driving behavior despite substantially reducing the DecisionxR1 model size.

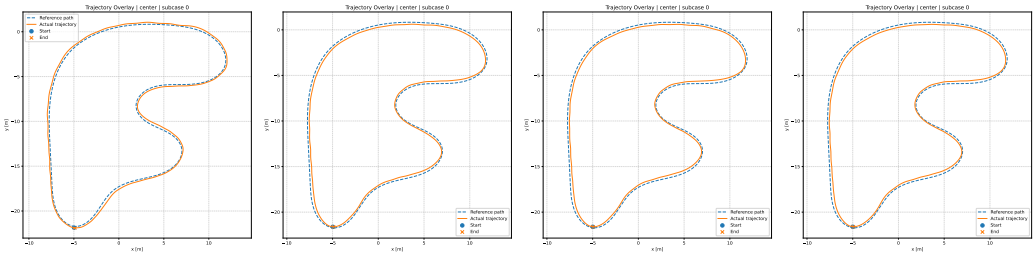


Figure 10: Qualitative ROS trajectories for DecisionxR1 pruned at 10%, 20%, 30%, and 45% with BPF-SFT/RL, keeping MPCxR1 unpruned.

C.4 Deployment on Multiple Reference Devices

Table 2 extends the deployment results of Sec. 4 by reporting both prefill and decode throughput on two devices: the Jetson AGX Orin onboard the robotic platform and an RTX A5000 workstation GPU. All models use the same deployment pipeline described in Sec. 4: BPF-SFT/RL-pruned checkpoints are quantized to `Q5_K_M`, exported to `GGUF`, and executed with `llama.cpp`. As in the main text, the dense baselines are re-measured on our deployment stack.

Table 2: Deployment metrics for dense and BPF-SFT/RL-pruned models on an RTX A5000 workstation GPU and on the Jetson AGX Orin onboard the robotic platform.

Module	Configuration	Memory (GB)	RTX A5000		Jetson AGX Orin	
			Prefill tok/s	Decode tok/s	Prefill tok/s	Decode tok/s
	Baseline 3B	2.07	7303	188	1744	45
	Baseline 1.5B	1.04	10865	270	3088	73
Decision	BPF-10%	1.92	7514	194	1863	49
	BPF-20%	1.77	8233	207	1894	51
	BPF-30%	1.62	7903	209	2073	56
	BPF-45%	1.51	9525	225	2184	58
MPC	BPF-10%	1.92	7890	197	1784	48
	BPF-20%	1.77	7092	200	1985	51
	BPF-30%	1.62	8262	210	2124	55