Robotics: Science and Systems 2026
Sydney, Australia, July 13–17, 2026

Vec-QMDP: Vectorized POMDP Planning on CPUs for Real-Time Autonomous Driving

Xuanjin Jin*†, Yanxin Dong*, Bin Sun‡, Huan Xu‡, Zhihui Hao‡, XianPeng Lang‡, Panpan Cai*†
*Shanghai Jiao Tong University, {xuanjin.jin, cai_panpan}@sjtu.edu.cn, yanxinn.d@gmail.com
†Shanghai Innovation Institute
‡Li Auto Inc., {sunbin1, xuhuan5, haozhihui1, langxianpeng}@lixiang.com
https://sii-boluomonster.github.io/VecQMDP-website

***Abstract*—Planning under uncertainty for real-world robotics tasks, such as autonomous driving, requires reasoning in enormous high-dimensional belief spaces, rendering the problem computationally intensive. While parallelization offers scalability, existing hybrid CPU-GPU solvers face critical bottlenecks due to host-device synchronization latency and branch divergence on SIMT architectures, limiting their utility for real-time planning and hindering real-robot deployment. We present Vec-QMDP, a CPU-native parallel planner that aligns POMDP search with modern CPUs' SIMD architecture, achieving $227\times$–$1073\times$ speedup over state-of-the-art serial planners. Vec-QMDP adopts a Data-Oriented Design (DOD), refactoring scattered, pointer-based data structures into contiguous, cache-efficient memory layouts. We further introduce a hierarchical parallelism scheme: distributing sub-trees across independent CPU cores and SIMD lanes, enabling fully vectorized tree expansion and collision checking. Efficiency is maximized with the help of UCB load balancing across trees and a vectorized STR-tree for coarse-level collision checking. Evaluated on large-scale autonomous driving benchmarks, Vec-QMDP achieves state-of-the-art planning performance with millisecond-level latency, establishing CPUs as a high-performance computing platform for large-scale planning under uncertainty.**

I. Introduction

Planning under uncertainty for real-world robotics tasks, such as autonomous driving, requires reasoning in enormous high-dimensional belief spaces. In dense urban settings, an autonomous vehicle must navigate interactive traffic flows where surrounding agents exhibit unknown intentions and behaviors. This uncertainty, compounded by perception noise, makes robust planning computationally intensive. Although Partially Observable Markov Decision Processes (POMDPs) provide a principled framework for such problems, the "curse of dimensionality" and "curse of history" [27] render existing solvers computationally intractable for real-time deployment on edge computing platforms, such as the onboard systems of autonomous vehicles.

While parallelization offers a pathway to scalability, existing solvers face critical bottlenecks on modern hardware. Hybrid CPU-GPU planners, such as HyP-DESPOT [5], suffer from significant host-device synchronization latency. Alternatively, GPU-native solvers like VOPP [16] are constrained by branch divergence on Single Instruction, Multiple Threads (SIMT) architectures. In autonomous driving, divergence occurs when different scenarios in a belief necessitate distinct logic—for instance, one branch may require yielding while another executes an overtake. This forces the hardware to serialize execution paths, severely degrading throughput and hindering high-frequency, closed-loop control.

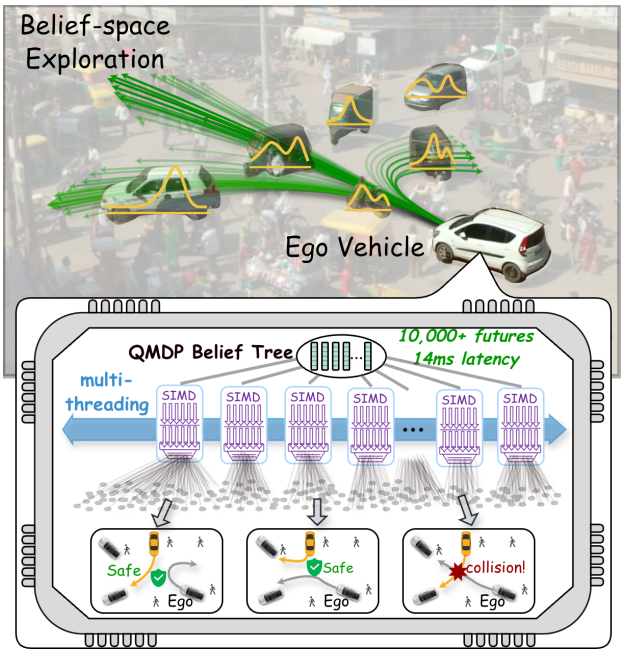


Fig. 1: Real-time belief-tree search in complex urban environments. Vec-QMDP achieves millisecond-level planning by parallelizing belief-tree search over $10{,}000+$ futures, enabling the ego-vehicle to navigate interactive traffic and respond to high-risk interactions within $14\,\mathrm{ms}$.

CPU-based Single Instruction, Multiple Data (SIMD) vectorization is another powerful parallelization paradigm. Recent works like VAMP [29] have proven it to be highly efficient, achieving up to $500\times$ speedups over single-threaded planning in sampling-based motion planning. However, parallelizing POMDP planning with SIMD is fundamentally more challenging. First, the uncertain future renders many different "scenarios" or future worlds, each producing a "scenario tree". Thus, the solver needs to consider many trees, not one, whose outcomes together determine the optimal plan. Second, the dynamics in large-scale urban environments are complex; multi-agent interaction renders non-linear trajectories, and the computation logic to simulate these dynamics is intricate. Thus, it is difficult to vectorize the dynamics function internally. Third,

collision checking is performed for a dynamic environment with many moving agents. Spatial trees, typically used in broad-phase checking, become different across scenarios and time steps, making it impossible to pre-compile them. Due to the above, vectorizing POMDP planning is hard. It requires a novel formulation that exploits the inherent multi-scenario structure of planning and the multi-agent nature of the urban driving problem.

To address the above challenges, we propose Vec-QMDP (Fig. 1), a parallel POMDP planner that combines CPU multi-threading and SIMD vectorization. It operates solely on the CPU, thus avoiding the communication overhead with GPUs. The planner is grounded in the QMDP approximation [24], which decomposes the belief tree into independent sub-trees after the initial action branching, allowing us to distribute the sub-trees across different CPU cores and SIMD lanes. To vectorize complex dynamics and collision checking, we introduce two modes of vectorization: *global vectorization*, i.e., batching the transition dynamics across nodes from different sub-trees to parallelize the forward simulation of different scenarios; and *local vectorization*, i.e., batching the agents within a single node to parallelize their collision test with the ego-vehicle, including the broad-phase spatial tree traversal and narrow-phase tests. Furthermore, to balance workloads across SIMD lanes during global vectorization, we employ a load-balancing Upper Confidence Bound (UCB) to synchronize the expansion-depth range of concurrent sub-trees, thus maximizing parallelism.

We evaluate Vec-QMDP on the large-scale nuPlan benchmark [3]. In terms of computational efficiency, our framework achieves $227\times$–$1073\times$ speedups in tree construction throughput, defined as the tree size constructed within unit planning time, compared to a state-of-the-art serial POMDP planner. This massive search capacity enables Vec-QMDP to deliver state-of-the-art driving performance with millisecond-level planning time, outperforming state-of-the-art learning models without requiring any training data.

II. Related Work

This section reviews efforts to accelerate tree-search-based planning under uncertainty, with an emphasis on how parallelization strategies interact with modern CPU/GPU execution models. We focus on three representative directions: CPU multi-threading for Monte Carlo tree search (MCTS), GPU parallelization for online POMDP planning, and CPU SIMD parallelization inspired by data-oriented layouts.

A. CPU Multi-threading for MCTS-based Planning

Parallel MCTS commonly uses root, tree, or leaf parallelization [7]. While effective when per-simulation cost is roughly uniform, many robotic tasks violate this assumption. In interactive robotics such as autonomous driving, simulation cost is highly non-uniform, so different threads often traverse different tree paths with unequal workloads, leading to load imbalance and frequent synchronization. Consequently, existing multi-threaded MCTS planners [7, 20, 6, 2] typically scale sub-linearly as CPU cores increase in continuous domains.

B. GPU Parallelization for Online POMDP Planning

GPUs offer high throughput and have motivated both hybrid CPU–GPU pipelines and GPU-resident solvers for online POMDP planning. HyP-DESPOT [5] accelerates planning by offloading large batches of Monte Carlo simulations to the GPU while retaining higher-level logic-heavy search on the CPU. This partitioning can improve device-side simulation throughput. However, end-to-end planning time may be constrained by host–device communication and by the synchronization required to integrate batched simulation returns across parallel search paths.

More recently, GPU-native planners such as VOPP [16] avoid host–device communication by keeping data structures and belief tree search on the GPU. However, online POMDP planning in interactive environments often contains data-dependent branching and thus exhibits irregular control flow. This irregularity manifests as *control-flow divergence* on SIMT hardware: when threads within a warp take different branches, the warp executes these paths sequentially via masking, reducing effective utilization and throughput [14].

C. CPU SIMD Parallelization for Tree-search-based Planning

While modern CPUs provide wide SIMD units, traditional Object-Oriented Programming (OOP) [11] layouts are often incompatible with them. Pointer-based structures fragment memory, reduce locality, and break the contiguous access patterns needed for efficient SIMD execution. Recent work such as VAMP [29] applies Data-Oriented Design (DOD) [13] to sampling-based motion planning in largely static scenes. In particular, Structure of Arrays (SoA) [28] stores each field contiguously across elements, enabling vectorized loads/stores and higher SIMD utilization.

Extending SIMD vectorization to online POMDP planning is harder. ROP-RAS3 [23] leverages VAMP-style SIMD acceleration to sample diverse macro-actions, improving scalability toward extremely long horizons. In contrast, Vec-QMDP targets real-time autonomous driving with a comprehensive CPU-native DOD framework that vectorizes the core planning cycle, including belief-tree expansion, dynamics simulation, and collision checking. This setting is especially challenging because the planner must evaluate many scenario trees rather than a single rollout. Urban driving also involves complex multi-agent interactions, causing data-dependent branching in the transition model, while dynamic collision checking requires spatial trees that vary across scenarios and time steps and therefore cannot be pre-compiled.

Vec-QMDP tightly couples multi-threading with SIMD under the QMDP approximation [24]. It applies *global vectorization* to batch transition dynamics across scenarios and *local vectorization* to accelerate within-node multi-agent collision checks, achieving up to a $1073\times$ speedup over state-of-the-art serial solvers.

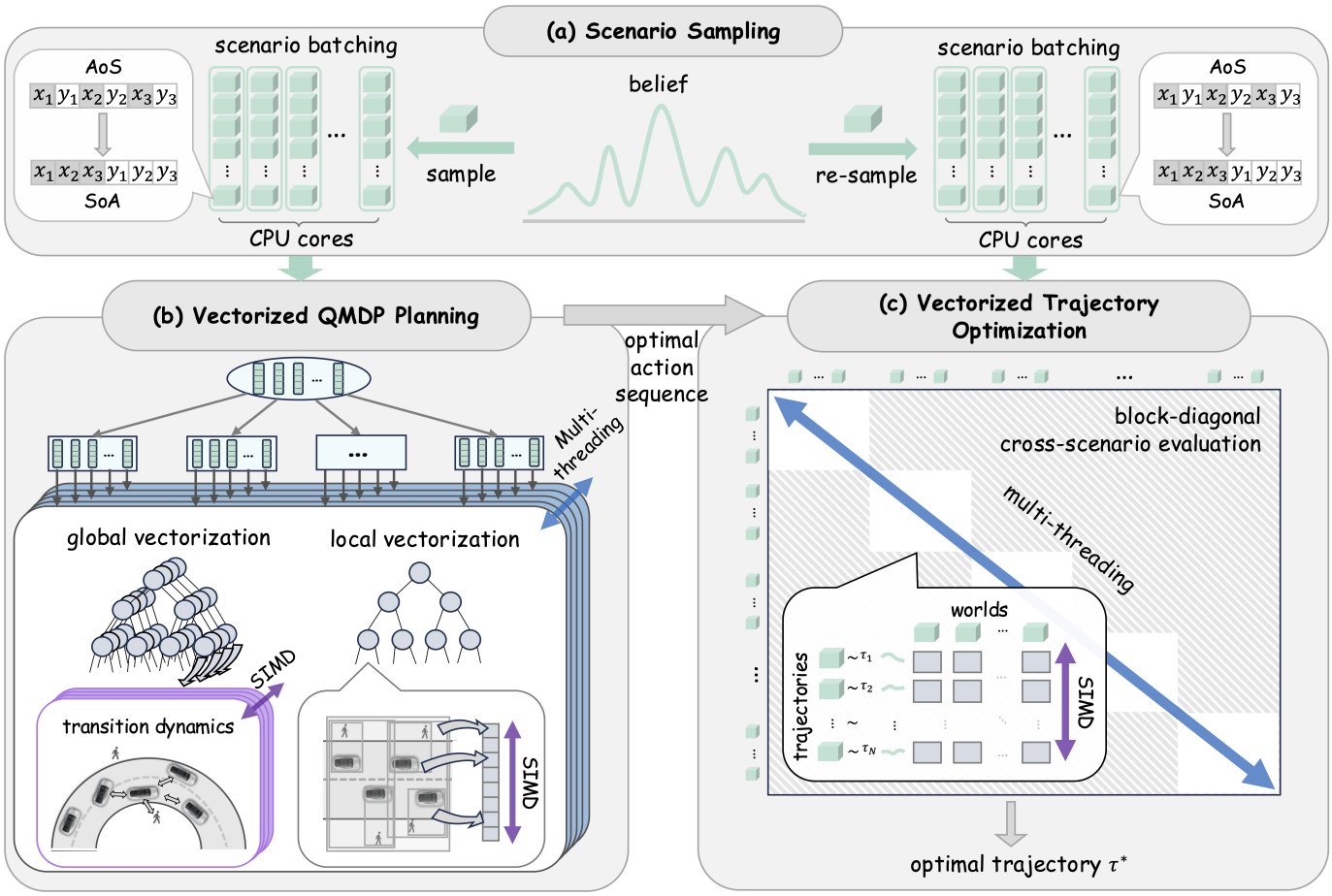


Fig. 2: Overview of Vec-QMDP. (a) Sample the belief into $M \times N$ scenarios in an SoA layout. (b) Vectorized QMDP search: after the first action, scenario trees run in parallel on M CPU threads; within each thread, SIMD *global vectorization* batches transition dynamics across scenarios and SIMD *local vectorization* accelerates within-node collision checks. (c) Vectorized trajectory optimization: generate candidates and use block-diagonal cross-scenario evaluation within minibatches to select τ^*.

III. Overview

Vec-QMDP (Fig. 2) scales up a state-of-the-art POMDP planner Hi-Drive [18] for autonomous driving by leveraging SIMD parallelism, demonstrating how belief tree search and belief-space trajectory optimization can be extensively vectorized for robotics tasks in complex dynamic environments. We build our parallel planner upon the QMDP approximation, which decomposes the belief tree into a set of independent sub-trees after the initial action branching, where each sub-tree corresponds to a sampled scenario of the future world. This decomposition allows us to distribute minibatches of scenario trees across different CPU cores for coarse-grained parallelization.

We then vectorize the QMDP belief tree search (Fig. 2b) using two distinct modes. *Global vectorization* batches search components across different scenario trees and assigns them to different SIMD lanes for parallel execution. This mode is applied to components with complex logic that are hard to vectorize internally, such as the transition dynamics of multi-agent interactions. *Local vectorization* vectorizes the internal computation of a search component. This mode is applied to components with clear factorization structure, such as collision checking between the ego-vehicle and independent exogenous agents. To support these vectorization modes, we refactor our data structures and algorithmic logic according to Data-Oriented Design (DOD) principles. We transform pointer-based trees into *vectorized trees* to enable SIMD-based traversal and convert Arrays of Structures (AoS) into Structures of Arrays (SoA) to ensure contiguous memory access.

Following the belief tree search, Vec-QMDP employs belief-space trajectory optimization (Fig. 2c) to refine driving trajectories based on the optimal action sequence. The core computation involves cross-scenario evaluation, where the ego-vehicle trajectory generated in each scenario is evaluated against the external world in other scenarios. We organize this evaluation as a block-diagonal sparse matrix, where each block concerns a minibatch of scenarios processed in a vectorized SIMD manner. Here, the vectorization is predominantly global, as the transition dynamics and collision checking of different scenarios are collated and vectorized.

The remainder of this paper is organized as follows: Sec-

tion IV formulates the POMDP model and the QMDP approximation. Section V details our vectorized QMDP planner. Section VI introduces our vectorized belief-space trajectory optimization. Section VII presents experimental results the large-scale nuPlan benchmark [3] and discussions.

IV. Mathematical Foundation for Parallelization

This section formalizes the POMDP model for autonomous driving and introduces the QMDP approximation [24], which provides the mathematical foundation for parallelization.

A. POMDP Model

We model driving as a POMDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{Z}, \mathcal{R}, b_0, \gamma \rangle$. A state $s \in \mathcal{S}$ includes the physical states of all agents (ego and exogenous), such as positions, speeds, headings, geometry, etc. Uncertainty arises from the behavioral intentions of exogenous agents, represented as predicted future trajectories $\boldsymbol{\xi} = \{\xi^i\}_{i=1}^{n_{\text{exo}}}$ on the planning horizon. We maintain a belief $b(\boldsymbol{\xi} \mid s)$ from a multimodal trajectory predictor (e.g., QCNet [35]), capturing multiple plausible future worlds conditioned on the current states.

The ego-vehicle selects a macro-action $a \in \mathcal{A}$ from a discrete set of reference paths with lateral nudges. Concretely, we use 3 candidate reference paths (e.g., lane-following, adjacent-lane change) and apply three lateral nudges of -1 m, 0 m, and +1 m to each path, yielding $|\mathcal{A}| = 9$ macro-actions. Each macro-action spans $\Delta t = 2$s with a finite planning horizon $T = 8$s. Conditioned on sampled multi-agent trajectory realizations $\boldsymbol{\xi}$, the transition advances the joint scene state by multi-agent simulation: $s' \sim \mathcal{T}(s' \mid s, a; \boldsymbol{\xi})$. Observations are noisy sensor readings modeled by $o \sim \mathcal{Z}(o \mid s')$, and the reward $\mathcal{R}(s, a)$ trades off safety, efficiency, and comfort with discount $\gamma \in [0, 1]$.

B. QMDP Approximation

To support real-time planning, Vec-QMDP adopts the QMDP approximation [24], which assumes uncertainty is resolved after the *first* step. We sample K scenarios $\phi = (s, \boldsymbol{\xi})$ from the current belief, where each scenario specifies a sampled initial state and determinizes the future evolution of exogenous agents, transitions, and observations. Conditioned on ϕ, the problem reduces to a fully observable MDP, and we compute the value function $V_\phi(\cdot)$ by lookahead search over the corresponding scenario tree.

Under QMDP, the root action value under belief b is approximated by averaging the post-action MDP values across K fixed scenarios, where $s'_k = \mathcal{T}(s, a; \boldsymbol{\xi}_k)$ is deterministic given $(s, a, \boldsymbol{\xi}_k)$:

$$Q_{\text{QMDP}}(b, a) \approx \frac{1}{K} \sum_{k=1}^{K} \left[R(s, a) + \gamma V_{\phi_k}(s'_k) \right], \tag{1}$$

$$a^* = \arg\max_{a \in \mathcal{A}} Q_{\text{QMDP}}(b, a). \tag{2}$$

This reveals the key parallel structure: after the first action branching, the K scenario trees become independent and can be solved in parallel. The resulting values are then aggregated to determine the optimal root action a^*. We recover the optimal action sequence π^* by following the highest-value action branches within the scenario trees. Vec-QMDP leverages this structure for hierarchical parallelization in Section V and refines π^* using belief-space trajectory optimization in Section VI.

Algorithm 1 Vectorized QMDP Belief Tree Search

Require: Initial belief b_0, number of CPU threads M, scenarios per thread N
Ensure: Optimal action sequence π^*

$\Phi = \{\phi_1, ..., \phi_K\} \leftarrow$ SampleScenarios(b_0), where $K = MN$
Expand root over all $a \in \mathcal{A}$
Initialize scenario trees $\mathcal{F} = \{\mathcal{T}_{\text{sc}}^{(1)}, \ldots, \mathcal{T}_{\text{sc}}^{(K)}\}$ conditioned on Φ after first action branching
for all thread $m \in \{1, \ldots, M\}$ **in parallel do**
  $\mathcal{F}_m \leftarrow \{\mathcal{T}_{\text{sc}}^{(m-1)N+1}, \ldots, \mathcal{T}_{\text{sc}}^{mN}\}$
  while within planning time budget **do**
    Traverse$(\mathcal{F}_m)$
    $\mathcal{F}_m \leftarrow$ VectorizedExpansion$(\mathcal{F}_m)$
    VectorizedRollout$(\mathcal{F}_m)$
    BackUp$(\mathcal{F}_m)$
    if Converged(Q-values) **then**
      break
    end if
  end while
end for
return $\pi^* \leftarrow$ ExtractActionSequence$(\mathcal{F})$

V. Vectorized QMDP Belief Tree Search

This section presents a CPU-native realization of hierarchical parallelism for QMDP belief tree search. Building on the QMDP decomposition in (1), we first describe the belief tree search in Algorithm 1, and then show how its Expansion/Rollout and reward evaluation are implemented with SIMD vectorization.

A. QMDP Planning for Urban Driving

Algorithm 1 performs finite-horizon belief-tree search over the discrete macro-action set $\mathcal{A}$ under K sampled scenarios. It samples $\Phi = \{\phi_k\}_{k=1}^K$ from the initial belief b_0, expands the root over all $a \in \mathcal{A}$, and decomposes the belief tree into K independent scenario trees $\mathcal{F} = \{\mathcal{T}_{\text{sc}}^{(1)}, \ldots, \mathcal{T}_{\text{sc}}^{(K)}\}$, each conditioned on one ϕ_k. The search then repeats Traverse to select a node, applies Expansion to execute one macro-action, runs Rollout to simulate to horizon H, and performs BackUp to update $Q(v, a)$, $N(v, a)$, and the *expansion-depth range* required by the load-balancing UCB in Section V-E. Planning stops when the root Q-values converge or the time budget is reached, and extracts π^* from the aggregated root Q-values under QMDP.

Expansion/Rollout simulates the joint evolution conditioned on the sampled $\boldsymbol{\xi}$. Exogenous agents deterministically follow the sampled trajectories $\boldsymbol{\xi}$, while the ego-vehicle executes a macro-action by following its reference path with

closed-loop interaction: longitudinal behavior follows Intelligent Driver Model (IDM) and lateral control uses a Stanley controller [31, 30], with lane changes gated by a Minimizing Overall Braking Induced by Lane changes (MOBIL) feasibility check [19]. This simulation is further accelerated in Section V-C.

Rewards are safety-dominated and evaluated against the same $\boldsymbol{\xi}$. For broad-phase filtering, we build a Frenet-based Sort-Tile-Recursive (STR)-tree [22] from predicted agent geometries at each time step and reuse it during search since $\boldsymbol{\xi}$ is fixed at the root. Broad-phase queries test the ego Frenet-frame Axis-Aligned Bounding Box (AABB) [4] against STR-tree node AABBs to prune candidates; leaf hits return a compact set of agent indices. We then gather the corresponding Cartesian states and apply a narrow-phase Separating Axis Theorem (SAT) test [17] to compute collisions, penalties, and terminal conditions. Frenet-frame AABBs align with road heading, reducing bounding-box inflation on curved roads and thus false positives. This reward pipeline is SIMD-vectorized in Section V-D.

B. Data-Oriented Representations for Vectorized Search

To enable SIMD processing, we refactor two pointer-based structures into SIMD-friendly layouts: the scenario search trees and the spatial STR-tree. Both have *fixed* maximum sizes, so we pre-allocate them as array-based balanced hierarchies with padding for absent nodes. For scenario trees, the capacity is fixed by planning depth H and branching $|\mathcal{A}|$, yielding N_{sc} (e.g., $N_{\mathrm{sc}} = \sum_{d=0}^{H} |\mathcal{A}|^d$). For STR-trees, the maximum leaf count is bounded by the number of exogenous agents n_{exo} and a fixed branching factor, yielding N_{str}. Each CPU thread owns a disjoint subset of scenarios and STR-trees, so all structures are built *intra-thread* without locks.

For each scenario tree, we flatten nodes into contiguous indices $v \in \{0, 1, \ldots, N_{\mathrm{sc}}-1\}$ and encode topology implicitly with fixed branching. For an action index $i \in \{1, \ldots, |\mathcal{A}|\}$, the i-th child of node v is addressed as $|\mathcal{A}| \cdot v + i$. We use a Structure-of-Arrays (SoA) layout for per-node statistics and flags, including Q-values, visit counts, immediate reward, and terminal/expanded markers. We also cache the *ego state* at each node, i.e., the kinematic state reached after executing the macro-action prefix along the root-to-v path. Concretely, ego states are stored as SoA vectors over nodes (e.g., $\{x^v\}_{v=0}^{N_{\mathrm{sc}}-1}$) Exogenous trajectories are scenario-level inputs fixed at the root under QMDP, so we store them once per scenario in a time-major layout, using SoA over agents at each time step (e.g., for each t, $\{x_t^i\}_{i=1}^{n_{\mathrm{exo}}}$).

The STR-tree is stored as an array-based balanced hierarchy in contiguous buffers, again pre-allocated with padding. We store AABB fields in SoA form (e.g., $\min_s, \max_s, \min_d, \max_d$ in the Frenet frame), so SIMD lanes can load multiple child boxes and test them against the ego AABB in parallel during broad-phase filtering.

C. Global Vectorization Across Scenario Trees

Global vectorization accelerates EXPANSION/ROLLOUT *across scenarios* by batching N expanded nodes drawn from N independent scenario trees within each CPU thread. Given the selected expanded node indices $\{v_i\}_{i=1}^{N}$, we *SIMD-gather* the corresponding ego states from SoA buffers into SIMD vector registers (e.g., $\{x^{v_i}\}_{i=1}^{N}, \{y^{v_i}\}_{i=1}^{N}$), together with per-node metadata (depth, terminal flags) and the macro-action IDs. Exogenous trajectories are scenario-level inputs fixed at the root, so each SIMD lane reads aligned per-time-step agent states from a time-major layout, with an SoA organization over agents at each time step.

In VECTORIZEDEXPANSION, we execute the selected macro-action for all N SIMD lanes using a single step-synchronous kernel. At each simulation step, the kernel advances all SIMD lanes in lockstep by loading the required per-SIMD-lane ego states, evaluating per-SIMD-lane control updates with masked execution, and writes back the updated ego states to the corresponding node buffers. The control flow remains largely uniform due to the macro-action design, with divergence occurring only when a macro-action triggers a lane change. In this case, only the affected SIMD lanes perform a MOBIL path-change feasibility check, while others are masked.

VECTORIZEDROLLOUT reuses the same step-synchronous kernel and differs only in termination. SIMD lanes are progressively masked once they reach horizon H or trigger a terminal event, and the rollout continues until all SIMD lanes are masked. Since expanded nodes can start at different depths (*depth divergence*), SIMD lanes may terminate at different times while the SIMD group runs to the last active SIMD lane, motivating load-balanced selection in Section V-E.

D. Local Vectorization Within a Scenario Node

Local vectorization accelerates reward evaluation by SIMD-parallelizing collision checking *within* a scenario node, as illustrated in Fig. 3. We batch multiple exogenous agents into one SIMD group and process them in lockstep.

In broad-phase filtering (Fig. 3a), STR-tree traversal is SIMD-accelerated at each expansion/rollout step. For an intersected internal node, we *batch-load* child AABBs from SoA buffers into SIMD vector registers and perform vectorized AABB intersection tests against the ego-vehicle's AABB. We descend only into intersecting children; leaf hits return candidate agent indices, which we then *SIMD-gather* into SIMD vector registers as compact blocks of exogenous states (e.g., position, heading, geometry) for refinement.

In narrow-phase refinement (Fig. 3b), we apply a SIMD Separating Axis Theorem (SAT) kernel, where each SIMD lane evaluates one ego–agent pair. We use masking with *early exit*: a SIMD lane stops once a separating axis is found, and the kernel terminates when all SIMD lanes exit or all SAT axes are tested. The kernel outputs collision flags and contact outcomes for rewards and termination.

E. Selection with Load-Balancing UCB

To mitigate the load imbalance noted in Section V-C, we modify TRAVERSE (Alg. 1, line 7) to reduce depth divergence across scenario trees in different SIMD lanes.

Each node s maintains an expansion-depth range $\mathcal{D}(s) = [d_{\min}(s), d_{\max}(s)]$, where $d_{\min}$ and $d_{\max}$ represent the minimum and maximum depths of all unexpanded expanded nodes in the subtree rooted at s. A new node initializes $\mathcal{D}(s)$ based on its own depth. During BACKUP (Alg. 1, line 10), $\mathcal{D}(s)$ is updated by taking the union of the current node's depth range and the depth ranges of its child nodes, terminating early if no change occurs.

Within each SIMD batch, selection proceeds in two stages. We first run standard UCB [1] independently in each scenario tree to obtain a tentative expanded node and record its depth; a majority vote over these depths defines the reference depth d_{ref}. We then re-run selection using a load-balancing UCB. For an action a at node s with child s', we score:

$$\mathcal{U}(s,a) = \text{UCB}(s,a) - \lambda \cdot |\text{clamp}(d_{\text{ref}}, \mathcal{D}(s')) - d_{\text{ref}}|\,,$$

where larger λ increasingly biases selection toward children whose expansion-depth range contains (or is closest to) d_{ref}, aligning selected depths across scenario trees for better load balancing.

VI. VECTORIZED BELIEF-SPACE TRAJECTORY OPTIMIZATION

Leveraging the optimal action sequence from the belief tree search, Vec-QMDP refines a continuous trajectory that is robust under scenario uncertainty.

A. Belief-space Trajectory Optimization with Importance Sampling

To focus optimization on safety-critical interactions, we resample K scenarios using importance sampling [25]. Risk-relevant agents are identified from map topology and predicted intersections between their future trajectories and the ego-vehicle's reference path. A proposal distribution q shifts probability mass of these critical agents toward hazardous intentions, while the nominal belief b is retained for non-critical agents.

For each sampled scenario, a candidate ego trajectory is generated by forward simulation under the action sequence, following the transition dynamics described in Section V-A. Robust evaluation requires *cross-scenario evaluation*: for each trajectory–scenario pair (τ_k, ϕ_i), we simulate the world forward conditioned on executing τ_k and evaluate the resulting state sequence using the reward model from Section V-A, yielding $V(\tau_k \mid \phi_i)$. The expected value of a trajectory is estimated via self-normalized importance sampling [15]:

$$E[V(\tau_k)] = \frac{1}{\sum_{i=1}^{K} w_i} \sum_{i=1}^{K} w_i\, V(\tau_k \mid \phi_i),$$
$$\tau^* = \arg\max_{\tau_k} E[V(\tau_k)],$$

where $w_i = \prod_j \frac{b(\xi_{i,j})}{q(\xi_{i,j})}$ is the product of importance weights over all agents j in scenario ϕ_i.

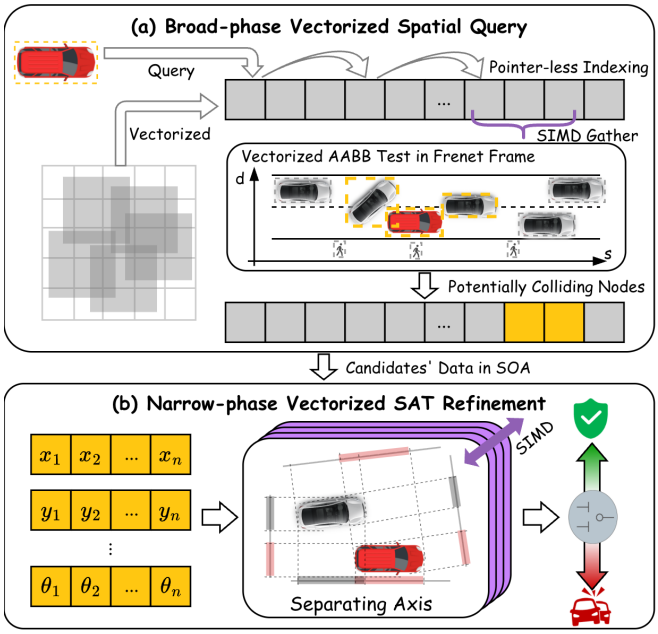


Fig. 3: Two-stage SIMD collision checking. (a) Broad phase: SIMD AABB tests in the Frenet frame traverse a pointer-less STR-tree to prune candidates. (b) Narrow phase: SIMD SAT checks evaluate ego–agent pairs to compute collisions.

B. Parallel Block-Diagonal Cross-Scenario Evaluation

A full cross-scenario evaluation scales as $O(K^2)$ and is intractable in real time. To mitigate this, we partition the K scenarios across M CPU threads, where each thread handles a minibatch Φ_m of N scenarios, restricting cross-evaluation within the minibatch and reducing per-thread complexity to $O(N^2)$.

Within each thread, computation is organized around *global vectorization* across trajectories. Ego trajectories for all N scenarios in Φ_m are generated in parallel via SIMD-batched forward simulation, as described in Section V-C. During evaluation, trajectory points across this SIMD batch are processed together: each trajectory is evaluated across all N scenarios in Φ_m, assessing their respective future states. Reward computation is dominated by collision checking. For each time step, the batched ego geometries form a query region that is intersected with the scenario's STR-tree to obtain candidate agents. Vectorized narrow-phase checks are then performed, with each check comparing multiple ego trajectories to each candidate exogenous agent. This structure enables efficient parallelization across both ego trajectories and exogenous agents during collision checking, making belief-space optimization tractable on CPU platforms.

VII. EXPERIMENTS

We evaluate Vec-QMDP on the large-scale nuPlan benchmark [3] to validate its performance in complex, interactive traffic scenarios under strict real-time constraints. Our results demonstrate that Vec-QMDP achieves competitive driving performance, with planning times as low as 9ms and peak scores within a 14ms planning time. In terms of computational efficiency, Vec-QMDP delivers a 227×–1073× speedup in

TABLE I
DRIVING PERFORMANCE COMPARISON ON NUPLAN.

Type	Planner	Val14		Test14-random		Test14-hard		Inference / Planning Time (ms) ↓
		R	NR	R	NR	R	NR	
Expert	Log-replay	80.32	93.53	75.86	94.03	68.80	85.96	-
Learning-based	PLUTO	78.11	88.89	78.62	89.90	59.74	70.03	-
	Diffusion Planner	82.86±0.14	89.67±0.05	84.60±0.16	89.71±0.32	68.77±0.15	75.30±0.31	80
Hybrid	PDM-Hybrid	92.14±0.00	92.77±0.00	91.79±0.00	94.56±0.00	76.12±0.00	66.09±0.00	171
	PLUTO w/ refine.	76.88	92.88	90.29	92.23	76.88	80.08	-
	Diff. Planner w/ refine.	92.90	<u>94.26</u>	91.75	<u>94.80</u>	82.00	78.87	>80
Model-based	Hi-Drive	93.01±0.12	93.22±0.08	91.95±0.13	93.46±0.31	<u>83.07</u>±0.18	81.22±0.31	503
	Hyp-DESPOT	93.07±0.06	93.26±0.13	91.83±0.15	93.69±0.20	82.83±0.29	81.82±0.28	259
	Vec-QMDP (match, Ours)	<u>93.15</u>±0.11	94.16±0.03	<u>92.51</u>±0.00	**95.21**±0.00	**84.23**±0.35	<u>82.30</u>±0.48	9
	Vec-QMDP (best, Ours)	**93.22**±0.06	**94.36**±0.02	**93.04**±0.05	**95.21**±0.00	**84.23**±0.35	**82.84**±0.11	14

* ***Bold*** *indicates best;* <u>underscored</u> *indicates second-best.*
Entries without confidence intervals are reported as published in the corresponding original papers.

belief tree construction throughput compared to the state-of-the-art serial POMDP planner Hi-Drive, with the gains scaling proportionally to scene complexity. While the evaluation focuses on autonomous driving, the core principles of CPU-parallelized belief tree search—leveraging multi-threading and SIMD to handle high-dimensional belief spaces and complex multi-agent dynamics—are applicable to other robotics domains requiring similar reasoning capabilities.

A. Experimental Setting

We evaluate Vec-QMDP on the nuPlan benchmark [3] using three subsets of increasing difficulty: *Val14* [12] (1,118 regular scenes), *Test14-random* [10] (268 random scenes), and *Test14-hard* [10] (272 complex scenes). Each scene involves a 15-second closed-loop simulation, evaluated in both Non-Reactive (NR) (static log-replay) and Reactive (R) (interactive agents) modes. All test scenes are directly taken from nuPlan without modification; dense scenes with large numbers of agents naturally arise from urban traffic involving vehicles, pedestrians, and cyclists.

Baselines include contrastive imitation learning (PLUTO [9]), diffusion-based planning (Diffusion Planner [34]), hybrid optimization (PDM-Hybrid [12]), the state-of-the-art serial POMDP planner Hi-Drive [18], and HyP-DESPOT [5], a CPU multi-threaded parallel variant of Hi-Drive. All methods are evaluated under identical hardware settings. Reported runtimes measure planning only and exclude prediction and perception, since nuPlan provides processed geometric and kinematic states. For a fair comparison among POMDP planners, Hi-Drive, HyP-DESPOT, and Vec-QMDP use the same $K = 64$ sampled scenarios, and both HyP-DESPOT and Vec-QMDP are run with 8 CPU threads.

Experiments are conducted on an Intel Xeon Gold 6530 CPU, where Vec-QMDP uses AVX2 with SIMD width $N = 8$. The implementation also supports AVX-512 on x86 and NEON on ARM platforms, making the proposed Data-Oriented Design and SIMD vectorization transferable to embedded automotive SoCs such as NVIDIA Orin. Prior SIMD results on Jetson AGX Orin suggest an approximate $2.2\times$ slowdown relative to our AVX2 setting [32], indicating strong potential for lightweight onboard deployment.

B. Evaluation Metrics

We evaluate planning quality and computational performance using the following metrics:

Driving Score and Safety Metrics: We use the standard nuPlan scoring function to measure closed-loop driving performance, which aggregates safety, progress, and rule-compliance indicators. In addition to the overall Driving Score (DS), we report key safety-related nuPlan metric scores: Safety Score (SS), corresponding to the not-at-fault collision score; TTC Score (TTCS), which evaluates whether the minimum time-to-collision satisfies the prescribed safety bound; and Drivable Area Compliance Score (DACS), which evaluates whether the ego vehicle remains within the mapped drivable area.

Tree Construction Throughput: To quantify computational capacity, we define *tree construction throughput* as the number of constructed tree edges per unit planning time (edges/ms). Each expanded edge corresponds to one *future*, i.e., one simulated transition branch under a sampled scenario; thus, expanding a node at depth d with maximum horizon H contributes $H-d$ futures. For a fair comparison, both Hi-Drive and HyP-DESPOT use the same QCNet-predicted trajectories as Vec-QMDP, and their edge counts are weighted by the number of scenarios visiting each edge. Since the per-edge processing time in Vec-QMDP is stable across the tree (mean 0.038 ms, std. 0.013 ms), each edge can be treated as an atomic unit, making edges/ms a robust measure of real-time search capacity.

TABLE II
SAFETY PERFORMANCE COMPARISON ON NUPLAN.

Method	Val14 (NR)			Test14-Hard (NR)		
	SS↑	TTCS↑	DACS↑	SS↑	TTCS↑	DACS↑
HyP-DESPOT	98.74	90.79	100.00	96.32	83.46	98.90
Vec-QMDP	99.36	93.91	100.00	96.76	83.79	98.16

C. Comparison on Driving Performance

As shown in Table I, Vec-QMDP outperforms learning-based, model-based, and hybrid baselines across all nuPlan scenes. We report two planning-time budgets to assess the efficiency–performance trade-off.

Vec-QMDP (match) achieves performance parity with state-of-the-art methods within only 9 ms, while Vec-QMDP (best) reaches peak driving performance within 14 ms. At the 14 ms budget, runtime is dominated by expansion, rollout, and trajectory optimization (3.23 ms, 4.05 ms, 4.68 ms), while other components remain lightweight, including scenario sampling (0.86 ms), STR-tree construction (0.99 ms), selection (0.28 ms), and backup (0.17 ms). Compared with HyP-DESPOT, Vec-QMDP achieves 1683.49 versus 10.88 edges/ms, corresponding to a 154.73× average tree-construction-throughput speedup. The safety results in Table II further show that Vec-QMDP improves Safety Score and TTC Score over HyP-DESPOT on both Val14 and Test14-hard, suggesting that the QMDP approximation preserves safety relative to full POMDP planning. Including QCNet prediction latency, Vec-QMDP still achieves an end-to-end latency of 65 ms, lower than Diffusion Planner's 80 ms, while obtaining better driving performance.

Performance gains are most pronounced on Test14-hard. In these challenging scenes, the high tree-construction throughput enables Vec-QMDP to evaluate many futures within 14 ms, allowing it to detect and negotiate low-probability, high-risk interactions, such as aggressive cut-ins, that are difficult for lower-throughput planners to capture.

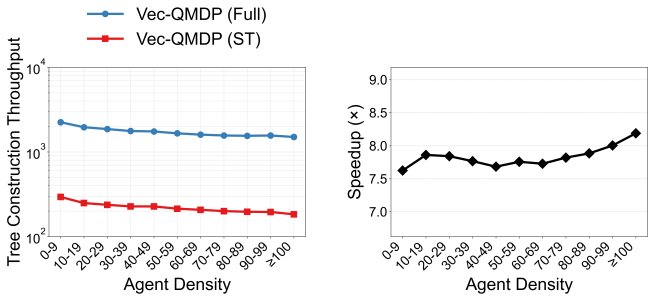
D. Computational Throughput Comparison

Vec-QMDP's computational efficiency is evaluated by analyzing tree construction throughput across varying traffic densities. Fig. 5 shows that while the serial baseline Hi-Drive declines sharply with agent density, Vec-QMDP maintains robust throughput. The observed speedup, scaling from 227× in sparse settings to 1073× in dense traffic, is a result of multi-tiered parallelization, which overcomes computational bottlenecks in dense traffic, enabling real-time planning.

E. Ablation Study on Multi-threading

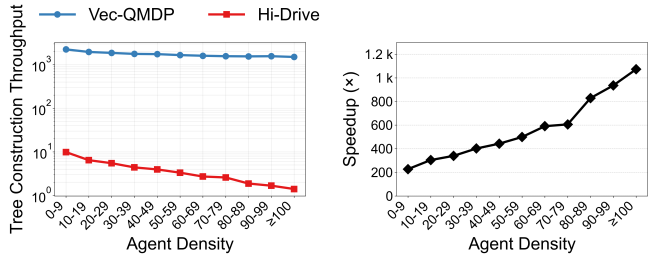
We isolate the contribution of thread-level parallelism by comparing our full multi-threaded implementation ($M = 8$ cores) with a single-threaded variant (ST). Both use identical SIMD-vectorized kernels and Data-Oriented Design (DOD).

Fig. 4 shows that while tree construction throughput declines gradually as agent density rises, the performance gap remains consistent across densities. The speedup remains near



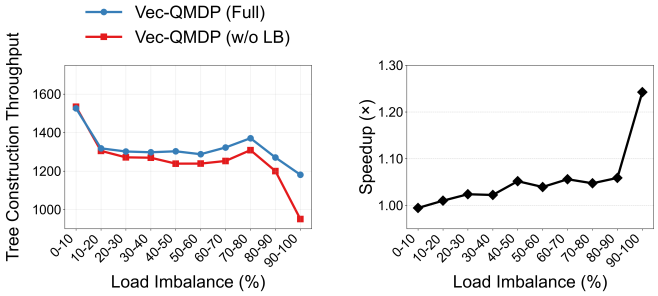
(a) Throughput (edges/ms) **(b)** Speedup (Full vs. ST)

Fig. 4: Ablation: multi-threading. (Left) Edges/ms vs. traffic density. (Right) Speedup over single-threaded (ST), showing near-linear scaling (∼8×).



(a) Throughput (edges/ms) **(b)** Speedup over Hi-Drive

Fig. 5: Tree construction throughput. (Left) Edges/ms vs. traffic density. (Right) Speedup over serial Hi-Drive (227×–1073×), increasing with density.



(a) Throughput (edges/ms) **(b)** Speedup (w/ LB vs. w/o LB)

Fig. 6: Ablation: load-balancing UCB. (Left) Edges/ms vs. load imbalance. (Right) Speedup over w/o LB, up to 1.24× in highly imbalanced scenes.

8×, indicating nearly linear scaling with core count. This confirms that our lock-free architecture eliminates synchronization overhead, enabling Vec-QMDP 's search throughput to scale with available CPU cores.

F. Ablation Study on Load-Balancing UCB

We ablate the proposed load-balancing UCB (Sec. V-E) by comparing the full planner (w/ LB) against a UCB-only variant (w/o LB).

Load imbalance metric. In the w/o-LB variant, we run standard UCB independently across scenario trees and obtain the expansion-depth ranges d_i. We define the depth difference as $\Delta d = \max_i d_i - \min_i d_i$, and measure *load imbalance* as

$$\text{Imbalance} = \frac{\#\{\text{simulations with } \Delta d \geq 1\}}{\#\{\text{total simulations}\}},$$

TABLE III
ABLATION STUDY OF LOAD BALANCING ON NUPLAN.

	Val14 (NR)		**Test14-Hard** (NR)	
Method	DS↑	SS↑	DS↑	SS↑
w/o LB	94.32	99.31	82.83	96.98
w/ LB	94.36	99.36	82.84	96.76

i.e., the fraction of simulations whose expansion-depth ranges are misaligned across SIMD lanes. We compute this metric only from the w/o-LB run to capture the inherent depth divergence induced by plain UCB.

Fig. 6 shows that the throughput gain (w/ LB over w/o LB) increases monotonically with the imbalance metric. When imbalance exceeds 90%, load-balancing UCB improves throughput by $\sim 24\%$ by aligning selection depths via d_{ref} and $\mathcal{D}(\cdot)$, thereby reducing SIMD lane idling during vectorized expansion and rollout. As shown in Table III, this throughput improvement does not materially affect planning quality: driving scores remain slightly higher with LB, while safety scores remain comparable on both Val14 and Test14-hard.

G. Scalability Test on Planning Time

We analyze how driving performance scales with planning time on the nuPlan benchmark, varying the time limit from 0ms to 100ms. Fig. 7 shows that both driving score and tree size stabilize within approximately 14ms. This rapid convergence indicates that the planner identifies high-quality driving trajectory almost immediately, resolving complex multi-agent interactions well within a real-time planning. The constructed tree size also stabilizes quickly, reaching a plateau that reflects the effective coverage of the belief space required for safe navigation.

These results demonstrate that our parallel POMDP planner efficiently scales performance with incremental CPU compute, and achieves near-optimal performance at 14ms, well below the conventional 100ms planning cycle.

VIII. CONCLUSION

We introduced Vec-QMDP, a CPU-native framework for real-time POMDP planning that leverages hardware parallelism. By refactoring traditional pointer-based trees into vectorized structures using Data-Oriented Design (DOD), our approach combines multi-threading with SIMD vectorization. Through *global vectorization*, we batch transition dynamics across independent scenario trees, while *local vectorization* accelerates internal search components, such as collision checking. Additionally, load-balancing UCB aligns expansion-depth range across concurrent sub-trees to maximize SIMD lane utilization. Evaluation on the nuPlan benchmark demonstrates that Vec-QMDP achieves $227\times$–$1073\times$ speedups in tree construction throughput, delivering state-of-the-art driving performance with millisecond-level planning times.

The main limitation of Vec-QMDP is its reliance on the QMDP approximation, which assumes that uncertainty is resolved after the initial action and therefore cannot fully capture deep active information gathering. Nevertheless, this

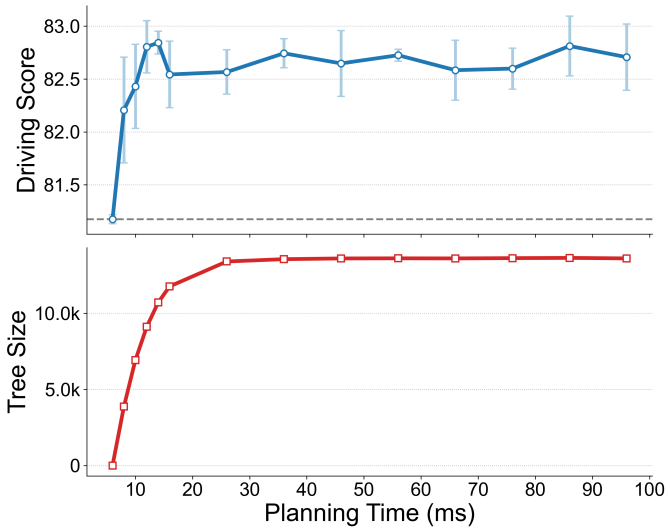


Fig. 7: Planning-time scalability. Driving score (top) and tree size (bottom) vs. planning time; both plateau by $\sim$14ms. Error bars: 95% CI.

assumption is well aligned with common autonomous-driving prediction pipelines, where a fixed set of future trajectories is generated before planning, while passive information gathering through future observations is still preserved. Moreover, when augmented at test time with an entropy-reduction reward following POMDP-lite [8], Vec-QMDP achieves higher cumulative reward than the advanced POMDP solver DESPOT [33] on RockSample(15,15) [26] (16.824 vs. 13.356) with a $157\times$ speedup (4.2 ms vs. 660.7 ms), suggesting that the practical loss of information-gathering capability can be partly mitigated. A second limitation is that SIMD efficiency depends on computational uniformity across scenario trees; highly heterogeneous dynamics may reduce lane utilization.

Future work will extend the proposed global and local vectorization principles beyond QMDP to more general online POMDP solvers such as DESPOT, enabling richer belief-dependent reasoning. Another promising direction is scaling Vec-QMDP to large or high-dimensional action spaces, for example by combining vectorized POMDP search with learned macro-actions [21] for high-DOF robotic systems.

IX. ACKNOWLEDGMENTS

This work is supported by New Generation Artificial Intelligence-National Science and Technology Major Project (No. 2025ZD0122901). This work was supported in part by the National Natural Science Foundation of China under Grant 62303304.

REFERENCES

[1] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

[2] Semanti Basu, Sreshtaa Rajesh, Kaiyu Zheng, Stefanie Tellex, and R Iris Bahar. Parallelizing pomcp to solve complex pomdps. In *Rss workshop on software tools for real-time optimal control*, 2021.

[3] Holger Caesar, Juraj Kabzan, Kok Seang Tan, Whye Kit Fong, Eric Wolff, Alex Lang, Luke Fletcher, Oscar Beijbom, and Sammy Omari. nuplan: A closed-loop ml-based planning benchmark for autonomous vehicles. *arXiv preprint arXiv:2106.11810*, 2021.

[4] Panpan Cai, Chandrasekaran Indhumathi, Yiyu Cai, Jianmin Zheng, Yi Gong, Teng Sam Lim, and Peng Wong. Collision detection using axis aligned bounding boxes. In *Simulations, Serious Games and Their Applications*, pages 1–14. Springer, 2013.

[5] Panpan Cai, Yuanfu Luo, David Hsu, and Wee Sun Lee. Hyp-despot: A hybrid parallel algorithm for online planning under uncertainty. *The International Journal of Robotics Research*, 40(2-3):558–573, 2021.

[6] Tristan Cazenave and Nicolas Jouandeau. On the parallelization of uct. In *Computer games workshop*, 2007.

[7] Guillaume MJ-B Chaslot, Mark HM Winands, and H Jaap van Den Herik. Parallel monte-carlo tree search. In *International Conference on Computers and Games*, pages 60–71. Springer, 2008.

[8] Min Chen, Emilio Frazzoli, David Hsu, and Wee Sun Lee. Pomdp-lite for robust robot planning under uncertainty. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5427–5433. IEEE, 2016.

[9] Jie Cheng, Yingbing Chen, and Qifeng Chen. PLUTO: Pushing the Limit of Imitation Learning-based Planning for Autonomous Driving, April 2024. arXiv:2404.14327.

[10] Jie Cheng, Yingbing Chen, Xiaodong Mei, Bowen Yang, Bo Li, and Ming Liu. Rethinking imitation-based planners for autonomous driving. In *2024 IEEE International Conference on Robotics and Automation*, pages 14123–14130. IEEE, 2024.

[11] Brad J Cox. Object oriented programming. 1984.

[12] Daniel Dauner, Marcel Hallgarten, Andreas Geiger, and Kashyap Chitta. Parting with misconceptions about learning-based vehicle motion planning. In *Conference on Robot Learning*, pages 1268–1281. PMLR, 2023.

[13] Richard Fabian. Data-oriented design. *Richard Fabian*, 2013.

[14] Tianyi David Han and Tarek S Abdelrahman. Reducing branch divergence in gpu programs. In *Proceedings of the fourth workshop on general purpose processing on graphics processing units*, pages 1–8, 2011.

[15] Timothy Classen Hesterberg. *Advances in importance sampling*. Stanford University, 1988.

[16] Marcus Hoerger, Muhammad Sudrajat, and Hanna Kurniawati. Vectorized online pomdp planning. *arXiv preprint arXiv:2510.27191*, 2025.

[17] Johnny Huynh. Separating axis theorem for oriented bounding boxes. pages 3–45, 2009. URL https://jkh.me/files/tutorials/Separating%20Axis%20Theorem%20for%20Oriented%20Bounding%20Boxes.pdf.

[18] Xuanjin Jin, Chendong Zeng, Shengfa Zhu, Chunxiao Liu, and Panpan Cai. Hi-drive: Hierarchical pomdp planning for safe autonomous driving in diverse urban environments. *IEEE Robotics and Automation Letters*, 2025.

[19] Arne Kesting, Martin Treiber, and Dirk Helbing. General lane-changing model mobil for car-following models. *Transportation Research Record*, 1999(1):86–94, 2007.

[20] Karl Kurzer, Christoph Hörtnagl, and J Marius Zöllner. Parallelization of monte carlo tree search in continuous domains. *arXiv preprint arXiv:2003.13741*, 2020.

[21] Yiyuan Lee, Panpan Cai, and David Hsu. MAGIC: Learning Macro-Actions for Online POMDP Planning . In *Proceedings of Robotics: Science and Systems*, Virtual, July 2021. doi: 10.15607/RSS.2021.XVII.041.

[22] Scott T Leutenegger, Mario A Lopez, and Jeffrey Edgington. Str: A simple and efficient algorithm for r-tree packing. In *Proceedings 13th international conference on data engineering*, pages 497–506. IEEE, 1997.

[23] Yuanchu Liang, Edward Kim, Wil Thomason, Zachary Kingston, Hanna Kurniawati, and Lydia E Kavraki. Scaling long-horizon online pomdp planning via rapid state space sampling. *arXiv preprint arXiv:2411.07032*, 2024.

[24] Michael L Littman, Anthony R Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In *Machine Learning Proceedings 1995*, pages 362–370. Elsevier, 1995.

[25] Yuanfu Luo, Haoyu Bai, David Hsu, and Wee Sun Lee. Importance sampling for online planning under uncertainty. *The International Journal of Robotics Research*, 38(2-3):162–181, 2019.

[26] Trey Smith and Reid Simmons. Heuristic search value iteration for pomdps. *arXiv preprint arXiv:1207.4166*, 2012.

[27] Adhiraj Somani, Nan Ye, David Hsu, and Wee Sun Lee. Despot: Online pomdp planning with regularization. *Advances in neural information processing systems*, 26, 2013.

[28] Robert Strzodka. Abstraction for aos and soa layout in c++. In *GPU computing gems Jade edition*, pages 429–441. Elsevier, 2012.

[29] Wil Thomason, Zachary Kingston, and Lydia E Kavraki. Motions in microseconds via vectorized sampling-based planning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8749–8756. IEEE, 2024.

[30] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, et al. Stanley: The robot that won the darpa grand challenge. *Journal of field Robotics*, 23(9):661–692, 2006.

[31] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical review E*, 62(2):1805, 2000.

[32] Jianyu Wei, Shijie Cao, Ting Cao, Lingxiao Ma, Lei Wang, Yanyong Zhang, and Mao Yang. T-mac: Cpu renaissance via table lookup for low-bit llm deployment on edge. In *Proceedings of the Twentieth European*

Conference on Computer Systems, pages 278–292, 2025.

[33] Nan Ye, Adhiraj Somani, David Hsu, and Wee Sun Lee. DESPOT: Online POMDP planning with regularization. *Journal of Artificial Intelligence Research*, 58:231–266, 2017.

[34] Yinan Zheng, Ruiming Liang, Kexin ZHENG, Jinliang Zheng, Liyuan Mao, Jianxiong Li, Weihao Gu, Rui Ai, Shengbo Eben Li, Xianyuan Zhan, and Jingjing Liu. Diffusion-based planning for autonomous driving with flexible guidance. In *The Thirteenth International Conference on Learning Representations*, 2025.

[35] Zikang Zhou, Jianping Wang, Yung-Hui Li, and Yu-Kai Huang. Query-centric trajectory prediction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 17863–17873, 2023.