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

Muninn: Your Trajectory Diffusion Model But Faster

Gokul Puthumanaillam*, Hao Jiang*, Ruben Hernandez*, Jose Fuentes†,
Paulo Padrao‡, Leonardo Bobadilla†, and Melkior Ornik*
*University of Illinois Urbana-Champaign. Email: {gokulp2, haoj5, rubenjh2, mornik}@illinois.edu
†Florida International University. Email: jfuen099@fiu.edu, bobadilla@cs.fiu.edu
‡Providence College. Email: ppadraol@providence.edu

Abstract*—Diffusion-based trajectory planners can synthesize rich, multimodal robot motions, but their iterative denoising makes online planning and control prohibitively slow. Existing accelerations either modify the sampler or compress the network–sacrificing plan quality or requiring retraining without accounting for downstream control risk. We address the problem of making diffusion-based trajectory planners fast enough for real-time robot use without retraining the model or sacrificing trajectory quality, and in a way that works across diverse state-space diffusion architectures. Our key insight is that diffusion trajectory planners expose two signals we can exploit: a cheap probe of how their internal trajectory representation changes across steps, and analytic coefficients that describe how denoiser errors affect the sampler's state update. By calibrating the first signal against the second on offline runs, we obtain a per-step score that upper-bounds how far the final trajectory can deviate when we reuse a cached denoiser output, and we treat this bound as an uncertainty budget that we can spend over the denoising process. Building on this insight, we present Muninn, a training-free caching wrapper that tracks this uncertainty budget during sampling and, at each diffusion step, chooses between reusing a cached denoiser output when the predicted deviation is small and recomputing the denoiser when it is not. Across standard benchmarks spanning offline RL planning (D4RL), configuration-space motion planning, and visuomotor diffusion policies, Muninn delivers up to 4.6× wall-clock speedups across several trajectory diffusion planners and diffusion policies by reducing denoiser evaluations, while preserving task performance and safety metrics. Muninn further certifies—at a user-chosen deviation tolerance and risk level—that cached rollouts remain within a specified distance of their full-compute counterparts, and we validate these gains in real-time closed-loop navigation and manipulation hardware deployments. Code, dataset and supplementary videos available at: https://github.com/gokulp01/Muninn.

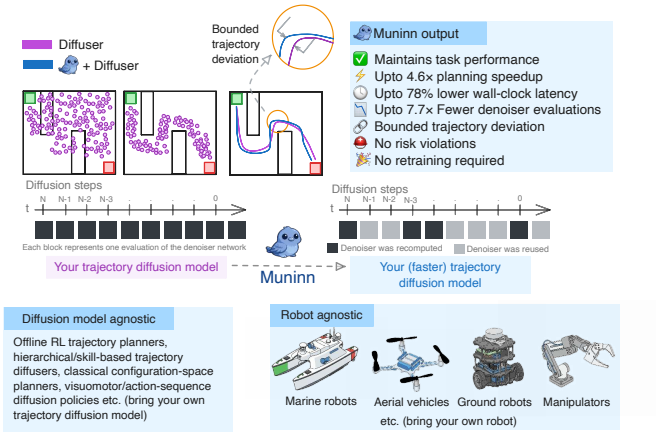


Fig. 1: High-level overview of Muninn applied to Diffuser. Diffuser recomputes the denoiser at every diffusion step, while Muninn wraps the same model and selectively reuses cached denoiser outputs at some steps, reducing compute while leaving the overall trajectory generation unchanged.

I. Introduction

Autonomous robots are increasingly steered by diffusion-based trajectory models [74, 65] that generate entire motion sequences [56, 67], capturing rich, multimodal futures [46] in cluttered and uncertain environments. These planners are typically run as fixed-depth reverse diffusion processes [45]: at every control cycle, the robot draws noise, rolls out dozens of denoising steps [17], and only then extracts a trajectory. On powerful servers this is acceptable; on embedded platforms or high-rate control loops, the cost is often prohibitive [64]. Simply truncating the sampler, shrinking the network, or running at a lower planning rate can help [33], but risks brittle failures: the model's behavior changes in opaque ways, and there is no clear link between "saved compute" and "lost plan quality." This paper, therefore, poses a precise question: how can we speed up diffusion-based trajectory planners by reusing internal computations, while keeping their behavior close to the original full-compute model and without retraining? A large body of work bypasses this issue by relying on model-based planning [41] and trajectory optimization [76, 54] with handcrafted costs and dynamics. These approaches can be fast [23], but tend to struggle when dynamics, human interactions, or multi-goal behaviors induce strongly multimodal solution sets [13, 28]. Diffusion-based planners address this [32] by learning complex trajectory distributions from data, yet they usually treat the denoiser as a black box and accept its inference cost as fixed. Efforts to reduce this cost follow several paths. One direction distills the diffusion process [59] into shallower networks [69] or coarse-to-fine samplers [68], but requires retraining for each architecture and sometimes each task. Another direction trains early-exit or policy networks [44] to decide when to stop denoising or which subnet to run; these methods can adapt computation per sample, but demand large-scale reinforcement learning or supervision [3] and tie the acceleration logic to a

specific model.

Recently, training-free accelerations [39, 27] have emerged that operate at inference time [4]. These approaches are attractive because they require no new data [36], works on existing architectures [7], and can deliver large wall-clock gains [72]. However, they are typically calibrated on perceptual or pixel-space metrics and developed for vision backbones. Little is known about how their approximation errors translate into trajectory deviations, and almost nothing is said about the consequences of those deviations for closed-loop control.

We make two observations about diffusion-based planners: (i) every denoising step passes the current noisy trajectory and diffusion timestep through a relatively cheap input-processing stage to produce a representation that feeds the deeper network blocks. Changes in this representation across timesteps provide a natural, low-cost signal for how stable the trajectory is becoming, (ii) the sampler used to map denoiser outputs into new trajectories has a known analytic form: for a given schedule, we can write down exactly how a perturbation in the predicted noise at step t changes the state at step $t-1$. Taken together, these two facts suggest that the planner itself contains enough structure to estimate, for each step, how risky it would be to reuse a past denoiser output instead of recomputing a new one. These observations motivate the central idea of this work.

Statement of contributions: We introduce (i) Muninn[1], a training-free wrapper for trajectory diffusion models that uses a cheap, timestep-wise probe of the internal representation to predict how much the final trajectory would change if a cached denoiser output were reused; (ii) a trajectory-level error budget that aggregates these predictions into decisions about when to reuse past denoiser outputs and when to recompute them, trading off planning speed against fidelity; and (iii) an evaluation on several diffusion-based planners across simulated and hardware navigation and manipulation benchmarks. We also release the code, dataset and models.

Note (scope). Diffusion planners span many architectures [74, 65]; we focus on *state-space* trajectory diffusion models that denoise future state or state–action sequences conditioned on observations and goals. Extending Muninn to other families mainly requires redefining the probe; the caching framework is unchanged, though the adaptation can be nontrivial.

II. Related work

Trajectory diffusion models: Early learning-based trajectory planners used imitation learning [50, 10] and variational generative models [25] to clone expert demonstrations. Conditional VAEs [26], normalizing flows [53], and graph latent models [19] capture diverse behaviors, but emit a single trajectory and rely on delicate training heuristics. Diffusion-based planners instead learn a denoising process over trajectories, achieving SOTA coverage of multimodal motions and have been applied to visual robot control [9], motion planning [21], and multi-agent behavior forecasting [75]. By sampling different noise realizations, these models naturally explore distinct homotopy classes [34] and avoid hand-crafted reward engineering. Yet standard samplers still require tens of denoising steps per plan [15], leading to second-level latencies and motivating recent work on accelerated trajectory diffusion [57].

Accelerating Diffusion via Model Design: In this vein, architectures compress the sampling chain while preserving multimodal coverage [58, 24]. One line of work distills multi-step samplers into shallow or one-step networks [52], or learns coarse-to-fine denoisers that jump directly to high-quality trajectories in a few iterations [16]. Other works bias the generative process with structured priors, trajectory libraries, or policy heads [14], so that sampling is guided toward task-feasible plans and fewer refinement steps are needed [18]. A complementary direction aggregates a set of diffusion rollouts into graphs, reusing evaluations across branches to reduce latency without sacrificing robustness [22]. These approaches demonstrate that careful parameterisation can shrink diffusion horizons, bringing trajectory diffusion closer to real-time operation [31, 55]. However, these accelerations still rely on sequential denoising, task-specific heuristics and typically operate by hard-coding new samplers or compressing the backbone network [49]. They degrade trajectory quality, require retraining for each architecture, and offer no guarantees on how the modified sampling process affects downstream control performance [42].

Adaptive Inference and Early Exiting: Beyond architectural changes, a parallel line of work accelerates diffusion models via dynamic inference [12], using learned early-exit policies [60] or step-skipping heuristics [61]to reduce test-time compute. In diffusion models, such methods typically monitor simple signals along the sampling chain to decide when to stop, skip, or jump ahead in the denoising schedule [43, 37, 73]. Related approaches in sequence modeling and control reuse cached network outputs across time or across similar inputs, treating repeated evaluations as an optimization lever rather than changing the underlying architecture [30, 1]. These strategies are usually trained in a task-specific, data-driven fashion and lack explicit bounds on how shortcutting or cache reuse perturbs the final sample [8, 38]. They provide little formal guidance on how much an accelerated diffusion run can deviate from its full-compute counterpart, and offer no guarantees on the quality of the executed trajectory [5, 40].

In contrast to prior works, we place Muninn at the intersection of trajectory diffusion and sampling-time acceleration, focusing on making existing trajectory planners fast enough without modifying or retraining their underlying networks.

III. Preliminaries and challenges

A pretrained diffusion planner operates over trajectories $\mathcal{T} \subseteq \mathbb{R}^{H\times n}$ and contexts $c \in C$ via a fixed reverse-diffusion sampler: $\tau_{t-1}^{\text{full}} = \Phi_t\big(\tau_t^{\text{full}},\ \varepsilon_\theta(\tau_t^{\text{full}}, t, c),\ \xi_t\big), t = T, \dots, 1$, where ε_θ is the denoiser and Φ_t is a known sampler update (DDPM (Denoising Diffusion Probabilistic Models)/DDIM (Denoising Diffusion Implicit Models)-style; ξ_t captures any sampler noise,

[1]"For Huginn I fear lest he come not home, / But for Muninn my care is more ..."– *Norse mythology*. Muninn is Odin's raven of *memory*. We borrow the name as a metaphor–relying on memory to move faster, while being careful not to stray far from the path we would have taken.

with $\xi_t \equiv 0$ for deterministic samplers). Let $\tau_0^{\text{full}}(c,\xi)$ be the resulting full-compute trajectory (evaluating ε_θ at every step). Each denoiser call has cost $\ell_t(c,\xi) \geq \ell_{\min} > 0$, so a full planning call costs $\mathcal{P}_c^{\text{full}}(c,\xi) = \sum_{t=1}^{T} \ell_t(c,\xi)$.

Challenges of caching in diffusion-based planners: The repeated denoising structure makes diffusion planners an appealing target for caching: the same network is invoked at every step, and consecutive timesteps often process very similar noisy trajectories. However, the same properties that make these models expressive, make safe caching surprisingly hard!

First, denoising is globally coupled over the horizon: each evaluation acts on the full trajectory τ_t, and the sampler mixes the predicted noise back into *all* time steps of τ_{t-1}, so a single approximation can bias the entire plan. Second, errors accumulate in a history-dependent way (see Fig. 2): reverse diffusion is iterative, so errors injected early are propagated and transformed by later updates; under nonlinear, contact-rich, or unstable dynamics, the same local denoiser error can have very different downstream effects depending on context and when it occurs. Third, control objectives are often discontinuous in trajectory space: small trajectory changes can flip outcomes from success to collision or constraint violation, so controlling only smooth surrogates can yield highly variable task impact.

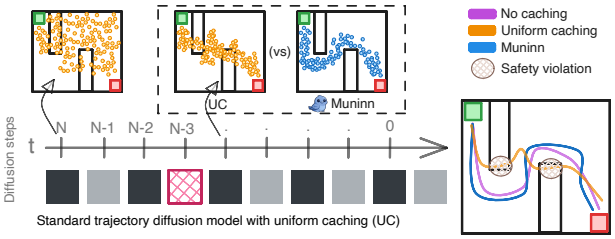


Fig. 2: Challenges in caching trajectory diffusion models. A mistaken reuse at one timestep (red) is not local: the reused output is mixed into all subsequent updates, causing errors to accumulate and shift the entire final trajectory.

Together, these effects create an all-or-nothing reuse regime: safe reuse requires confidence not only that the local approximation is small, but that its contribution remains acceptable after all subsequent sampler updates. Coarse strategies (e.g., fixed step skipping) oscillate between being overly conservative (limited speedup) and overly aggressive (hard-to-predict failures). Since practical diffusion planners vary in architecture, schedules, and conditioning, approaches that rely on retraining lose generality. This motivates caching mechanisms that (i) cheaply detect when reuse is risky, (ii) account for sampler-mediated error propagation, and (iii) expose interpretable knobs for trading compute against trajectory deviation.

IV. Muninn

Motivated by these challenges, Muninn is a *training-free* caching policy that, at each diffusion step, either recomputes $\varepsilon_\theta(\tilde{\tau}_t, t, c)$ or reuses a cached denoiser output to produce a lower-cost trajectory $\tilde{\tau}_0^{\pi}(c,\xi)$. It exposes two user-facing knobs: a deviation tolerance η_{traj} and risk level α, and aims to minimize expected planning subject to $\mathbb{P}\Big(d(\tau_0^{\text{full}}, \tilde{\tau}_0^{\pi}) > \eta_{\text{traj}}\Big) \leq \alpha$.

A. Trajectory diffusion and sampler sensitivity

We begin by making precise how local errors in the denoiser's output at each diffusion step affect the final trajectory.

Reverse diffusion as a paired process: Recall that $\tau_0^{\text{full}}(c,\xi)$ is produced by evaluating the denoiser at every step. Caching policies π modify this process by *reusing* denoiser outputs at selected timesteps instead of recomputing them. To isolate the effect of reuse, we analyze a paired run: the full and cached chains share the same initial sample τ_T and the sampler noise sequence $\xi = (\xi_T, \ldots, \xi_1)$, and differ only in the denoiser outputs Φ_t. Let $\tilde{\tau}_t$ denote the cached trajectory at step t and let $\tilde{\varepsilon}_t$ be the noise prediction actually used at step t (recomputed or reused). The cached chain evolves as $\tilde{\tau}_{t-1} := \Phi_t(\tilde{\tau}_t, \tilde{\varepsilon}_t, \xi_t)$. Muninn will choose $\tilde{\varepsilon}_t$ by *recomputing* $\tilde{\varepsilon}_t = \varepsilon_\theta(\tilde{\tau}_t, t, c)$ or by *reusing* a cached prediction.

Error variables: We define two per-step errors: the trajectory drift $\Delta_t := \tau_t^{\text{full}} - \tilde{\tau}_t$ and the denoiser mismatch $e_t := \varepsilon_\theta(\tilde{\tau}_t, t, c) - \tilde{\varepsilon}_t$., capturing divergence in trajectory space and in the noise prediction, respectively. Unless stated otherwise, $\|\cdot\|$ denotes the Frobenius norm on $\mathbb{R}^{H\times n}$. Our only assumption on the sampler is a local Lipschitz property:

Assumption 1 (Sampler Lipschitzness)**.** *For timestep t, there exist nonnegative constants K_t and L'_t such that, for any trajectories $\tau, \tau' \in \mathcal{T}$, noise predictions $\varepsilon, \varepsilon'$, and fixed sampler noise ξ_t, $\big\|\Phi_t(\tau,\varepsilon,\xi_t) - \Phi_t(\tau',\varepsilon',\xi_t)\big\| \leq K_t\|\tau-\tau'\| + L'_t\|\varepsilon-\varepsilon'\|$.*

For DDPM/DDIM-style samplers, Φ_t is affine in τ_t and ε_t, so K_t and L'_t follow directly from the update coefficients. Conceptually, K_t scales trajectory differences, while L'_t scales sensitivity to noise-prediction error. Applying Assumption 1 to the full-compute and cached updates at step t gives

$$\begin{aligned}\|\Delta_{t-1}\| &= \big\|\Phi_t(\tau_t^{\text{full}}, \varepsilon_\theta(\tau_t^{\text{full}}, t, c), \xi_t) - \Phi_t(\tilde{\tau}_t, \tilde{\varepsilon}_t, \xi_t)\big\| \\ &\leq K_t\|\tau_t^{\text{full}} - \tilde{\tau}_t\| + L'_t\big\|\varepsilon_\theta(\tau_t^{\text{full}}, t, c) - \tilde{\varepsilon}_t\big\| = K_t\|\Delta_t\| + L'_t\|e_t\|.\end{aligned} \tag{1}$$

This recursion decomposes the trajectory error at step $t-1$ into a part inherited from the existing trajectory mismatch Δ_t and a new contribution from the denoiser error e_t.

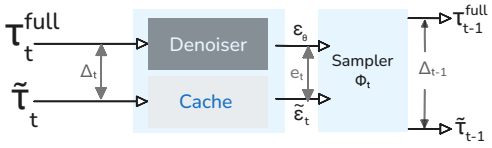


Fig. 3: Paired reverse-diffusion step. e_t is the denoiser error from reusing $\tilde{\varepsilon}_t$ and Δ_t is the induced trajectory mismatch; a Lipschitz bound splits Δ_{t-1} into propagated mismatch and injected error.

Unrolling the recursion: Both processes are initialized from the same noisy trajectory, so $\Delta_T = 0$. Repeatedly applying the local bound (1) from $t = T$ down to $t = 1$ yields a closed-form bound on Δ_0. For $t = T$ we have $\|\Delta_{T-1}\| \leq K_T\|\Delta_T\| + L'_T\|e_T\| = L'_T\|e_T\|$. Continuing, a simple induction shows,

$$\|\Delta_t\| \leq \sum_{s=t+1}^{T} \Big(L'_s \prod_{j=t+1}^{s-1} K_j\Big)\|e_s\| \quad \text{for } t \in \{0,\ldots,T-1\}. \tag{2}$$

Defining the pathwise sensitivity coefficients, $L_s := L'_s \prod_{j=1}^{s-1} K_j$, we can rewrite (2) at $t = 0$ compactly as

$$\|\Delta_0\| \;\le\; \sum_{t=1}^{T} L_t \|e_t\|. \tag{3}$$

Each step's denoiser error $|e_t|$ contributes additively to the final trajectory deviation, weighted by L_t, capturing how strongly errors at t are amplified through subsequent diffusion updates.

From norm difference to trajectory deviation: The guarantee in our problem statement is expressed in terms of a user-chosen trajectory metric $d : \mathcal{T} \times \mathcal{T} \to \mathbb{R}_{\ge 0}$. We assume that d is Lipschitz-equivalent to the norm $\|\cdot\|$ in the sense that:

Assumption 2 (Metric compatibility)**.** *There exists a constant* $\Gamma \ge 1$ *such that for all* $\tau, \tau' \in \mathcal{T}$, $d(\tau, \tau') \;\le\; \Gamma \|\tau - \tau'\|$.

Combining Assumption 2 with (3) yields:

$$d\big(\tau_0^{\text{full}}(c,\xi),\, \tilde{\tau}_0(c,\xi)\big) \;\le\; \Gamma\|\Delta_0\| \;\le\; \sum_{t=1}^{T} \Gamma L_t \|e_t\|. \tag{4}$$

Equation (4) is the key trajectory deviation bound. It exposes that an error e_t injected at step t is scaled by L_t, which depends on all later sampler sensitivities $\{K_j\}_{j<t}$. Early steps can therefore have a larger impact than late steps. In our experiments we use $d(\tau, \tau') := \frac{1}{\sqrt{H}}\|\tau - \tau'\|_F$, with $\Gamma = \frac{1}{\sqrt{H}}$.

We define the per-step trajectory cost associated with a $\|e_t\|$ as $c_t(e_t) := \Gamma L_t \|e_t\|$. Then (4) simply states that the total trajectory deviation is bounded by the sum of per-step costs,

$$d\big(\tau_0^{\text{full}}(c,\xi),\, \tilde{\tau}_0(c,\xi)\big) \;\le\; \sum_{t=1}^{T} c_t(e_t). \tag{5}$$

Muninn uses (5) as a budget rule: if each reuse step has a high-probability upper bound on $c_t(e_t)$ and the summed bounds stay within η_{traj}, then the final cached trajectory remains within η_{traj} of the full-compute one with the same high probability.

Now, the challenge is that, during deployment, the actual denoiser error $\|e_t\|$ is unknown for a reuse decision.

B. Probe features and per-step error scores

The sensitivity bound (4) quantifies how costly a step-t error $\|e_t\|$ would be, but not how large that error would be under reuse. In order to do that, we introduce (i) a lightweight *probe* F_t, and (ii) *per-step score* s_t. Fig. 4 summarizes this subsection.

Probe features: At step t, the denoiser $\varepsilon_\theta(\cdot, t, c)$ is evaluated on the current noisy τ_t and context c. In almost all modern architectures, this evaluation can be decomposed into: (i) an input processing or "stem" stage that maps (τ_t, t, c) into an intermediate representation h_t, and (ii) a deeper, expensive "core" that refines h_t into the final noise prediction $\varepsilon_\theta(\tau_t, t, c)$.

We exploit this decomposition by defining a probe feature F_t that depends only on the cheaper part of the computation and is therefore inexpensive to compute at every step.

Formally, we posit a probe function $\Psi : \mathcal{T} \times \{1, \dots, T\} \times C \to \mathbb{R}^{d_F}$ and define $F_t := \Psi(\tilde{\tau}_t, t, c)$. We require Ψ to satisfy two design goals: (i) The cost of evaluating Ψ at a single step should be negligible compared to a full pass through ε_θ. (ii) Architecture-agnostic: Any trajectory diffusion model provides an intermediate representation between its input and its final noise prediction. By defining Ψ in terms of this representation, we ensure that Muninn can be applied to a wide range of state-space diffusion planners without architectural changes.

Per-step error scores from probe dynamics: Intuitively, reuse is safe when the denoiser's output would not have changed much when recomputed. We expect this to happen when F_t has stabilized: if the probe at step t is similar to the nearby steps, the model is likely to produce similar noise predictions, and using a cached output should incur a small error $\|e_t\|$.

To capture this intuition, Muninn maps the sequence of probe features $\{F_t\}$ into a scalar *error score* $s_t \in \mathbb{R}_{\ge 0}$ at each step t. This score should be: (i) cheap to compute from probe features we have, (ii) monotonically informative in the sense that smaller scores tend to correspond to smaller denoiser errors.

Since reverse diffusion runs from $t = T$ to 1, the reuse decision at step t can use both the current probe F_t and the previously computed F_{t+1}. A effective choice of score is the magnitude of the probe change between these two steps:

$$s_t := \phi(F_t, F_{t+1}, t) = \frac{\|F_t - F_{t+1}\|_1}{\|F_{t+1}\|_1 + \omega}, \quad t \in \{1, \dots, T-1\}, \tag{6}$$

where $\omega > 0$. This design reflects two intuitions: (i) when h_t stabilizes, F_t changes slowly and the denoiser output changes little, so reuse yields small $\|e_t\|$; (ii) as sensitivity varies with timestep (via L_t), calibrating score-to-error bounds per t adapts reuse decisions to high-impact regions of the diffusion chain.

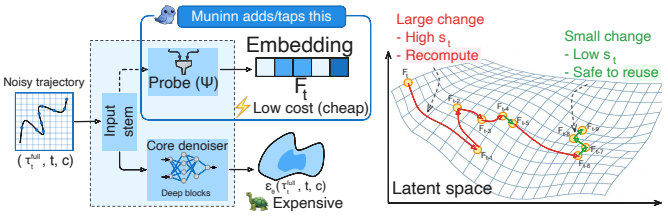


Fig. 4: Probe features and per-step error scores. (Left) A denoiser evaluation decomposes into a cheap stem and an expensive core; Muninn runs only the stem to compute a lightweight probe. **(Right)** Large step-to-step changes (s_t) trigger recomputation, while small s_t indicate stability and make reuse safe.

C. Conformal calibration of reuse error

Section IV-A bounds trajectory deviation by a sum of per-step costs. To use this bound online, Muninn needs a high-probability upper bound on the (unknown) reuse error $\|e_t\|$ at each step. We now show how to obtain these bounds.

Calibration data: Recall that the trajectory deviation bound (4) depends only on its magnitude $\|e_t\|$. We therefore define the scalar error $\epsilon_t := \|e_t\| \in \mathbb{R}_{\ge 0}$ and aim to predict ϵ_t from the score s_t introduced in (6).

At deployment, $\tilde{\varepsilon}_t$ is the noise prediction actually reused by Muninn at step t, which depends on the entire history of recompute vs. reuse decisions. For calibration, we fix a policy-agnostic reuse rule that defines a potential reuse output at every step, independent of the budget. We define a deterministic map that, specifies which earlier denoiser output would be reused

if we chose to reuse at t. For e.g., we can always reuse the prediction from the previous step that evaluated ε_θ.

Given this potential reuse rule, we build a calibration set by running the full-compute chain alongside a paired ghost reuse chain on N i.i.d. episodes from the deployment distribution. For each episode i and each $t \in \mathcal{T}_{\text{cache}}$, we record (i) the probe-based score $s_t^{(i)}$ computed from the ghost chain, and (ii) the corresponding potential reuse error magnitude $\epsilon_t^{(i)}$. The resulting calibration dataset is the collection of score–error pairs $\mathcal{S}_{\text{cal}} := \{(s_t^{(i)}, \epsilon_t^{(i)}) : i = 1, \dots, N,\ t \in \mathcal{T}_{\text{cache}}\}$, where $\mathcal{T}_{\text{cache}}$ is the set of timesteps at which reuse is allowed. Appendix E and Figure 5 provide the full construction details.

Since the reuse pattern is fixed and independent of s_t, we can view each $(s_t^{(i)}, \epsilon_t^{(i)})$ as a draw from a common, unknown joint distribution over scores and potential reuse errors induced by the base planner and environment. Under the exchangeability assumption—calibration and deployment episodes are sampled i.i.d. from $\mathcal{D}$—this is the setting where conformal prediction provides finite-sample coverage guarantees.

Split-conformal regression: Given $\mathcal{S}_{\text{cal}}$, we construct a function $U : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ that maps each score s to a high-probability upper bound on the corresponding potential reuse error ϵ. Concretely, we fit a regression model $m(s)$ on a training split of $\mathcal{S}_{\text{cal}}$ and apply split conformal prediction on a held-out split to obtain an additive residual quantile $q_{1-\alpha_{\text{step}}}$. This results in the bound $U(s) := \max\{m(s) + q_{1-\alpha_{\text{step}}}, 0\}$; Appendix E provides the full construction.

By split conformal prediction, assuming exchangeability between calibration and deployment, we obtain per-step coverage: for a new step with score s_t and potential reuse error ϵ_t,

$$\mathbb{P}\big(\epsilon_t \leq U(s_t)\big) \geq 1 - \alpha_{\text{step}}. \tag{7}$$

This guarantee is *distribution-free*: it holds for any joint distribution of (s_t, ϵ_t) as long as calibration and deployment samples are exchangeable. The regressor m only tightens U; even if m is misspecified, (7) holds by construction.

From per-step coverage to global risk: The per-step coverage (7) applies to the latent potential reuse error ϵ_t at any timestep t. Combining $c_t(e_t)$ with the upper bound $U_t(s_t)$ gives, for any step t, $\Gamma L_t \epsilon_t \leq \Gamma L_t U_t(s_t)$ whenever $\epsilon_t \leq U_t(s_t)$. If we define the upper-bounded per-step cost at score s_t as $\hat{c}_t(s_t) := \Gamma L_t U_t(s_t)$, then with probability at least $1 - \alpha_{\text{step}}$,

$$c_t(e_t) = \Gamma L_t \epsilon_t \leq \hat{c}_t(s_t). \tag{8}$$

Crucially, the event $\{\epsilon_t > U_t(s_t)\}$ depends only on the underlying data-generating process and the calibration procedure; it is independent of the caching policy we will deploy later.

D. Muninn caching policy

Muninn uses the per-step bounds $\hat{c}_t(s_t)$ to decide online, per sample, whether to reuse or recompute. The idea is to interpret (5) as a budget constraint: user specifies a maximum allowable η_{traj} which is spent across reverse-diffusion steps.

Budgeted reuse rule: At deployment time, the true error $\|e_t\|$ at t is unknown. Let $\mathcal{T}_{\text{cache}} \subseteq \{1, \dots, T\}$ denote the set of timesteps where reuse is permitted. We maintain a remaining budget B_{rem} and, for clarity, we summarize the algorithm below.

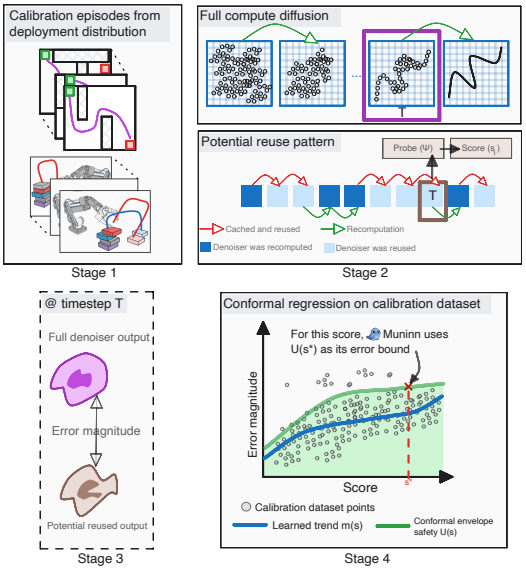


Fig. 5: (Stages 1–3) From offline rollouts, we build a calibration set of score–error pairs by running the full planner with a ghost reuse chain. **(Stage 4)** A regressor $m(s)$ models the typical score–error relationship, and split conformal prediction yields $U(s)$. At test time, Muninn maps each score to $U(s_t)$, converts it to a trajectory-level cost, and spends it from the budget.

Algorithm 1 Muninn caching policy for one planning call

Sample initial noisy trajectory $\tau_T \sim p_T$.
Initialize: (i) $B_{\text{rem}} \leftarrow \eta_{\text{traj}}$, (ii) $\tilde{\tau}_T \leftarrow \tau_T$.
for $t = T, T-1, \dots, 1$ **do**
  Compute probe $F_t = \Psi(\tilde{\tau}_t, t, c)$.
  if $t < T$ **then**
    Retrieve previous probe F_{t+1} // stored at step $t+1$.
    Compute score $s_t = \phi(F_t, F_{t+1}, t)$
    Compute upper-bounded cost $\hat{c}_t(s_t) = \Gamma L_t U_t(s_t)$.
  end if
  if $t = T$ **or** $t \notin \mathcal{T}_{\text{cache}}$ **or** $(t < T$ **and** $\hat{c}_t(s_t) > B_{\text{rem}})$ **then**
    Recompute:
      $\varepsilon_t^{\text{new}} \leftarrow \varepsilon_\theta(\tilde{\tau}_t, t, c)$
      Update cache with $(t, \varepsilon_t^{\text{new}})$ as most recent recompute.
      $\tilde{\varepsilon}_t \leftarrow \varepsilon_t^{\text{new}}$
  else
    Reuse:
      Set $\tilde{\varepsilon}_t$ to the cached denoiser output specified by the potential reuse rule.
      $B_{\text{rem}} \leftarrow B_{\text{rem}} - \hat{c}_t(s_t)$
  end if
  Apply sampler update: $\tilde{\tau}_{t-1} \leftarrow \Phi_t(\tilde{\tau}_t, \tilde{\varepsilon}_t, \xi_t)$
  Store F_t // for use at step $t-1$.
end for
return $\tilde{\tau}_0$

This policy is: (i) budget-safe by construction: enforcing $B_{\text{rem}} \geq 0$ ensures that the sum of upper-bounded costs over reuse steps satisfies $\sum_{t \in \mathcal{R}} \hat{c}_t(s_t) \leq \eta_{\text{traj}}$; (ii) sample-adaptive: reuse locations and counts depend on the realized scores s_t (and thus on the context, noise, and trajectory).

Global risk guarantee: Under the conformal calibration guarantee (7), for each t in a planning episode we have $\mathbb{P}\big(\epsilon_t \leq U_t(s_t)\big) \geq 1 - \alpha_{\text{step}}$. Define the failure event at step t as $\mathcal{F}_t := \{\epsilon_t > U_t(s_t)\}$. By construction of U_t,

$$\mathbb{P}(\mathcal{F}_t) \leq \alpha_{\text{step}} \quad \text{for each } t. \tag{9}$$

Recall that ϵ_t is defined w.r.t. the potential reuse rule, and at reuse, Muninn uses the same rule to choose which cached output to apply. Thus, whenever Muninn reuses at step t, the realized denoiser error magnitude $\|e_t\|$ coincides with the latent potential reuse error ϵ_t for that step.

Now consider a single planning episode. If none of the failure events $\mathcal{F}_t$ occur at any timestep in $\mathcal{T}_{\text{cache}}$, then for all reuse steps $t \in \mathcal{R}$ we have $c_t(e_t) = \Gamma L_t \epsilon_t \le \Gamma L_t U_t(s_t) = \hat{c}_t(s_t)$, and thus the total trajectory deviation satisfies

$$\begin{aligned} d(\tau_0^{\text{full}}, \tilde{\tau}_0) &\le \sum_{t \in \mathcal{R}} c_t(e_t) && \text{(from (5))} \\ &\le \sum_{t \in \mathcal{R}} \hat{c}_t(s_t) && \text{(since no failure event occurs)} \\ &\le \eta_{\text{traj}} && \text{(by the budget rule).} \end{aligned} \tag{10}$$

Therefore, the only way for the trajectory deviation to exceed η_{traj} is if at least one failure event occurs: $\{d(\tau_0^{\text{full}}, \tilde{\tau}_0) > \eta_{\text{traj}}\} \subseteq \bigcup_{t \in \mathcal{T}_{\text{cache}}} \mathcal{F}_t$. Taking probabilities and applying a union bound over timesteps yields

$$\begin{aligned} \mathbb{P}\big(d(\tau_0^{\text{full}}, \tilde{\tau}_0) > \eta_{\text{traj}}\big) &\le \mathbb{P}\Big(\bigcup_{t \in \mathcal{T}_{\text{cache}}} \mathcal{F}_t\Big) \\ &\le \sum_{t \in \mathcal{T}_{\text{cache}}} \mathbb{P}(\mathcal{F}_t) \quad \le |\mathcal{T}_{\text{cache}}|\, \alpha_{\text{step}}. \end{aligned} \tag{11}$$

If we choose $\alpha_{\text{step}} := \frac{\alpha}{|\mathcal{T}_{\text{cache}}|}$, then (11) simplifies to

$$\mathbb{P}\big(d(\tau_0^{\text{full}}, \tilde{\tau}_0) > \eta_{\text{traj}}\big) \;\le\; \alpha, \tag{12}$$

which is exactly the risk guarantee from the problem statement.

This guarantee does not require timestep independence: it follows from the per-step bounds (9) and a union bound. It holds for any adaptive reuse set $\mathcal{R}$, provided the reuse rule used to define ϵ_t in calibration matches the rule used at deployment.

E. Practical considerations and engineering choices

In practice, Muninn is implemented with a few simple engineering choices: (i) *forbidden regions*—we disallow reuse in a short prefix (large t, highly noisy and sensitivity-amplifying) and suffix (small t, where small changes in τ_t can directly affect executed motion); (ii) *batched sampling*—when multiple trajectories are denoised in parallel, we maintain an independent remaining budget B_{rem} per trajectory and apply the same reuse rule independently, so the guarantee (7) and bound (12) apply per trajectory; (iii) *model instantiations*—for Transformer trajectory denoisers we take Ψ as a short prefix of attention/MLP blocks and mean-pool tokens to obtain $F_t \in \mathbb{R}^{d_F}$, while for MLP denoisers we use the penultimate hidden layer as F_t, in all cases computing (F_t, s_t) *before* the reuse decision at step t and making them available during both calibration (to label $\|e_t\|$) and deployment (when $\|e_t\|$ is unknown).

V. Experiments

We evaluate Muninn on two trajectory-diffusion families covering the dominant uses of diffusion in robotics.

Trajectory diffusion planners (offline RL and motion planning) [6, 51, 21, 2, 62, 29, 35, 48] use diffusion as a planner over full state-action or configuration-space trajectories, implemented as an iterative reverse-diffusion process (optionally with guidance). Diffusion policies (imitation and visuomotor control) [9, 71] use diffusion as a policy that generates short-horizon action or pose segments, conditioned on high-dimensional observations, and executed repeatedly in a receding-horizon control loop.

A. Quantitative Results: Simulation and Benchmarks

We evaluate Muninn on trajectory centric benchmarks and report standard metrics as defined in the prior literature for each benchmark. See Appendix A for details on the environments, datasets, training, and metrics (Appendix C). All experiments are run on an NVIDIA A10 GPU (24GB VRAM) and averaged on 150 previously unseen episodes.

Offline RL / trajectory planning. We first consider the D4RL offline RL suite [11] for continuous control, where models act as trajectory optimizers/conditional policies over state-action.

Benchmark	Model	Task metr. ↑		Latency ↓		#Evals/t ↓		$\mathbb{E}[d]\downarrow$	$\hat{p}_{\text{viol}}\downarrow$
		Full	+🐇	Full	+🐇	Full	+🐇		
MuJoCo locomotion (D4RL)									
HalfCheetah 	Diffuser [21]	59.3	58.8	580	145	100	17.0	0.056	0.045
	Dec. Diff. [2]	62.9	61.9	116	40	20	5.4	0.042	0.038
	Diff.-QL [62]	63.7	63.6	20	10	10	4.4	0.030	0.028
	AdaptDiff[29]	59.6	59.5	353	101	60	12.4	0.050	0.041
Hopper 	Diffuser [21]	65.1	64.7	523	138	100	18.0	0.058	0.046
	Dec. Diff. [2]	70.6	69.8	105	39	20	6.0	0.045	0.040
	Diff.-QL [62]	75.7	75.6	20	11	10	4.7	0.032	0.030
	AdaptDiff[29]	63.8	63.6	310	97	60	14.2	0.052	0.043
Walker2d 	Diffuser [21]	77.8	77.2	783	224	100	20.6	0.062	0.048
	Dec. Diff. [2]	84.0	83.7	157	63	20	6.7	0.048	0.042
	Diff.-QL [62]	80.8	80.7	25	14	25	12.7	0.034	0.032
	AdaptDiff[29]	83.3	83.0	403	119	60	12.9	0.057	0.045
Goal-reaching navigation (D4RL)									
Maze2D 	Diffuser [21]	119.5	119.0	733	175	100	15.3	0.065	0.047
	Dec. Diff. [2]	130.8	129.6	147	46	20	4.7	0.055	0.044
	Diff.-QL [62]	105.6	104.8	25	12	10	4.2	0.040	0.035
	AdaptDiff[29]	144.3	143.9	347	99	60	12.4	0.060	0.046
	CompDiff[35]	160.2	159.2	1100	275	150	25.0	0.070	0.049
AntMaze 	Diffuser [21]	58.7	58.3	950	207	100	13.0	0.072	0.049
	Dec. Diff. [2]	72.2	71.8	190	53	20	4.0	0.060	0.046
	Diff.-QL [62]	59.6	59.4	25	10	25	8.8	0.045	0.038
	AdaptDiff[29]	63.0	62.7	550	145	60	10.9	0.065	0.047
	CompDiff[35]	75.9	74.5	1400	342	150	24.0	0.075	0.049
Long-horizon manipulation (D4RL)									
FrankaKitchen 	Diffuser [21]	53.4	53.0	800	235	100	21.6	0.058	0.045
	Dec. Diff. [2]	55.0	54.9	160	62	20	6.3	0.045	0.040
	AdaptDiff[29]	58.1	57.7	450	167	60	18.0	0.050	0.042
Robot block stacking (Kuka)									
Kuka stacking 	Diffuser [21]	52.3	51.8	1200	400	100	25.9	0.055	0.044
	Dec. Diff. [2]	60.4	60.0	240	100	20	7.0	0.045	0.040

TABLE I: Offline RL and trajectory planning. Task metric is the benchmark metric (D4RL normalized score for D4RL tasks; success rate for Kuka stacking). $d(\cdot,\cdot)$ is the trajectory deviation metric used for risk accounting and $\hat{p}_{\text{viol}} = \mathbb{P}\big(d(\tau_0^{\text{full}}, \tilde{\tau}_0) > \eta_{\text{traj}}\big)$.

Motion planning in configuration space. We then test Muninn

on classical motion planning problems where trajectories are represented in joint space rather than as RL episodes.

	Scen.	Model	Success(%)↑ Full		Collis. (%) ↓ Full		Lat. (ms) ↓ Full		$\hat{p}_{\text{viol}}$↓
Arm planning in clutter 	easy	MPD [6]	99.0	98.8	1.0	1.2	180	120	0.040
		EDMP[51]	99.5	99.4	0.5	0.6	120	92	0.035
	med.	MPD [6]	94.0	93.5	6.0	6.4	200	118	0.045
		EDMP[51]	96.5	96.2	3.5	3.7	135	95	0.040
	hard	MPD [6]	82.0	81.2	18.0	18.8	230	121	0.048
		EDMP[51]	88.0	87.5	12.0	12.4	160	107	0.044

TABLE II: Configuration-space motion planning. Scenarios correspond to increasing clutter/difficulty under the EDMP [51] evaluation protocol. Collision/violation includes obstacle collisions and any hard constraint violations.

Visuomotor imitation and manipulation. Finally, we evaluate Muninn on visuomotor policy diffusers that generate pose trajectories conditioned on images (2D) or 3D observations.

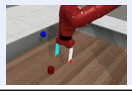
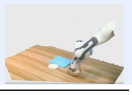
Benchmark/Metric		RLBench [20] Reach Target Diff. Policy [9] 	Meta-World[70] pick-place-v2 Diff. Policy [9] 	DP3 Pour DP3 [71] 
Success (%)↑	Full	99.0	92.0	98.0
		98.8	91.5	97.6
Latency (ms)↓	Full	25	30	45
		15	18	27
$\mathbb{E}[d]$ ↓		0.035	0.040	0.038
$\hat{p}_{\text{viol}}$ ↓		0.040	0.045	0.043

TABLE III: Visuomotor imitation and manipulation. Latency is measured per policy inference/control call. $d(\cdot,\cdot)$ is defined on the trajectory segment.

B. Quantitative Results: Hardware Evaluation

We further validate Muninn in real-world, previously unseen, closed-loop deployment on robotic platforms spanning navigation and manipulation. (i) *2D marine navigation:* a SeaRobotics Surveyor ASV performs waypoint-following in an open-water lake under partial observability. (ii) *3D aerial navigation:* a Crazyflie quadrotor performs 3D waypoint navigation. (iii) *Manipulation:* a SO-ARM100 tabletop arm executes pick-and-place, stacking, and peg insertion. Complete platform specifications are given in Appendix B.

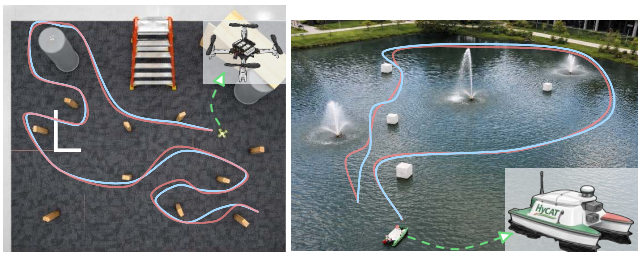


Fig. 6: Qualitative results. Full-compute trajectories (GC-Diffuser and B-COD teacher) are shown in red; Muninn-accelerated trajectories are shown in blue.

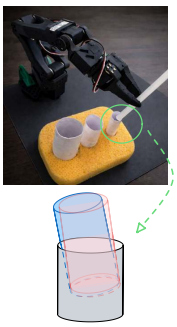


Fig. 7: Angle of attacks (hard peg insertion) Red: Diffusion Policy; blue: Muninn.

Pl.	Method	Lat. (ms) ↓	Succ. (%) ↑	Coll. (%) ↓
ASV	BCOD_{1s} [48]	12	94.0	2.4
	$\text{BCOD}_{tch.}$ [48]	60	97.0	1.6
	$\text{BCOD}_{tch.}$ +	24	96.8	1.7
	GC-Diff. [21]	70	95.5	2.2
	GC-Diff.+	28	95.3	2.3
UAV	GC-Diff. [21]	75	93.0	3.0
	GC-Diff.+	30	92.8	3.1
SO-100	Diff. policy [9]	40	82.0	6.0
	DP +	24	81.5	6.2

TABLE IV: We compare the Full (teacher) models against the baselines and Muninn on the waypoint navigation and manipulation (averaged over three tasks) tasks. (Average of 150 hardware runs).

C. Key findings, analysis, and ablations

As can be seen from the results in sections V-A and V-B, Muninn consistently reduces denoiser evaluations and wall-clock latency across all evaluated policies while preserving task performance. We now report key findings (KF) and summarize ablations of Muninn's internal design choices, with further breakdowns deferred to Appendix D.

(KF#1) Muninn dominates inference-time acceleration baselines on the *task–latency–risk* Pareto frontier. A common path to real-time diffusion planning is *training-free* inference-time acceleration, i.e., reducing denoiser work without retraining. Most such methods are compute-centric: they skip calls without modeling how errors propagate through the sampler or impact the final trajectory. We compare Muninn against three such baselines (Fig. 8, Table V): *(i)* FewSteps[47]: run the same pretrained denoiser with a reduced diffusion horizon $T' < T$. *(ii)* FixedSkip: evaluate the denoiser every k steps and reuse the last predicted noise for skipped steps. *(iii)* ProbeThresh: reuse if the probe-change s_t falls below a tuned threshold δ.

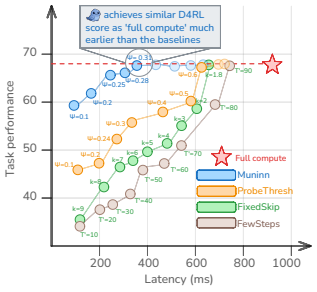


Fig. 8: We plot the D4RL score vs. wall-clock latency for D4RL AntMaze. Muninn achieves the same task success rate as Full model 2× quicker than the baselines.

Method	Task ↑	$\mathbb{E}[d]$↓	$\hat{p}_{\text{viol}}$↓
Full	68.7	0.00	0.00
FewSteps	56.1	0.12	0.17
FixedSkip	42.5	0.10	0.14
ProbeThresh	48.0	0.09	0.10
+ Full	67.6	0.07	0.04

TABLE V: We compare the Full (unmodified T-step sampler) model against the baselines and Muninn on the D4RL AntMaze task. The results above correspond to allowing Full to run for 957 ms, at which point it achieves the maximum possible D4RL score. The baselines and Muninn are capped at 250 ms (approximately 1/4th of the runtime).

Considering the offline RL domain (averaged across benchmarks in Table I), we find that Muninn yields a *strictly better* operating curve: at the same latency it preserves task success more reliably, and at the same $\hat{p}_{\text{viol}}$ it achieves lower latency. This happens because FewSteps and FixedSkip implicitly allocate approximation error uniformly in diffusion time, even though the sampler's sensitivity profile L_t is highly non-uniform. ProbeThresh adapts to probe dynamics, but without

calibrated error envelopes it violates risk in hard contexts or becomes overly conservative when tuned for safety. The baselines inherently produce small latent errors that become large deviations after downstream sampler mixing. Muninn is the only inference-time method in this set that (a) estimates reuse error with distribution-free coverage and (b) accounts for step-dependent error amplification through ΓL_t.

(KF #2) Muninn closes much of the gap to training-time accelerations *without retraining*. A second path to real-time diffusion planning is training-time acceleration. These methods can achieve very low latency, but require additional training compute, careful hyperparameter sweeps, and often task- or architecture-specific redesign. We compare Muninn (Fig. 9, Table VI) against: *(i)* DISTILL-1STEP/DISTILL-KSTEP[63, 66]: one/few-step distilled planners, *(ii)* SMALLNET:[66] a smaller denoiser trained from scratch or fine-tuned (fewer layers / width), *(iii)* LEARNEDEXIT[71]: a trained controller that predicts early stopping.

Method	Extra train.	Extra data	Arch. change
Full multi-step (teacher)	×	×	×
DISTILL-1STEP	▲	↻	✓
DISTILL-KSTEP (K=4)	▲	↻	✓
SMALLNET (0.6× width)	△	×	✓
LEARNEDEXIT	▲	✓	✓
Teacher +	×	×	×
DISTILL-KSTEP +	×	×	×

TABLE VI: Columns report whether each method requires extra training compute, additional data (beyond the original training set), or architectural changes for the DP3 Pour benchmark. We include DISTILL+Muninn to show composability. *Legend:* × = none, ✓ = yes, ↻ = often, △ = moderate, ▲ = high.

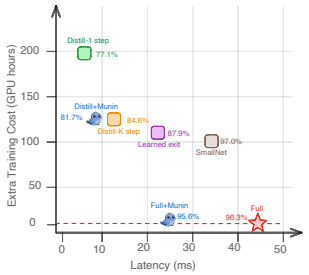


Fig. 9: Scatter plot summarizing the practical acceleration spectrum on the DP3 Pour diffusion policy. We also report the average success percentage of these models during inference. GPU-hours are normalized to A10 24GB-hours equivalent. More details about the training procedure is available in the Appendix.

We find that Muninn provides a deployment-friendly point on the spectrum: it recovers a large fraction of the achievable latency reduction *immediately*, while preserving the behavior of the original planner. Moreover, Muninn is composable: applying Muninn on top of a distilled or compact model yields additional speedups, because Muninn targets redundant per-step computation that remains even in fewer-step samplers.

(KF#3) Muninn induces *difficulty-adaptive compute*.

Eval bin	Fraction eps.	Success
10–12	0.20	0.99
13–15	0.30	0.97
16–18	0.25	0.94
19–21	0.15	0.90
22–25	0.10	0.87

Min. clearance	Avg. evals	Success
0.1	22	0.84
0.3	20	0.90
0.5	18	0.94
0.7	15	0.97
0.9	12	0.99

TABLE VII: Left: Episode fraction and Muninn success binned by #evaluations: low-eval bins are easier (higher success), high-eval bins are harder (lower success). **Right:** Compute and success vs. minimum-clearance difficulty: Muninn uses more evals. in harder and fewer in easier regions.

A hidden cost of fixed accelerations is that they spend the same compute on easy and hard situations. In robotics, difficulty varies across time (e.g., open-space navigation vs. tight corridors).

Muninn's decision rule is inherently state- and context-adaptive. Taking the example of cluttered 7-DoF arm planning (averaged across EDMP and MPD) (Table VII), we analyze the distribution of denoiser evaluations per episode and correlate it with environment difficulty proxies. The left panel in Table VII should be read as a distribution over Muninn's actual compute usage: episodes that require only 10–12 denoiser evaluations are typically easier and succeed more often, while episodes that consume 22–25 evaluations are harder and have lower success. The right panel Table VII makes this difficulty axis explicit using minimum clearance: as clearance decreases from open-space to tight-clutter settings, Muninn automatically recomputes more often instead of applying a fixed skip pattern. We find that Muninn naturally allocates more compute to high-risk or high-uncertainty regimes, and aggressively reuses outputs when the diffusion trajectory stabilizes.

(KF#4) Muninn's budget accounting produces a *usable runtime certificate* that correlates with downstream safety. Most acceleration methods are opaque: they reduce compute, but provide no introspection about when approximations become dangerous. Muninn is different because its internal quantity $\hat{D}$ is an explicit, online-tracked upper bound proxy (Fig. 10) for how far the cached trajectory might drift from full compute. This yields two practical benefits: *(i) Post-hoc explainability:* we can attribute most deviation risk to specific timesteps and contexts where L_t is large and the probe indicates instability. *(ii) Runtime escalation:* in a closed-loop controller, we can treat low remaining budget B_{rem} or large $\hat{D}$ as a trigger to (a) temporarily run full compute, (b) reduce the action magnitude, (c) switch to a conservative safety controller, or (d) request more samples. We present an extended study of the runtime escalations in the Appendix D.

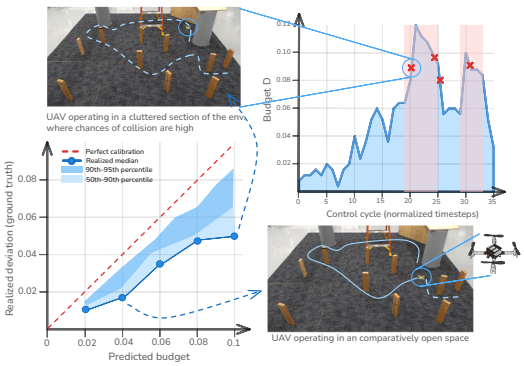


Fig. 10: Left: Reliability plot: predicted $\hat{D}$ vs. realized deviation d; Muninn is monotone and conservative (below the diagonal). **Right:** GC-Diffuser+Muninn on 3D UAV navigation: $\hat{D}$ over control cycles, with spikes aligning to near-collision/contact events (red); callouts show the UAV pose at those instants.

Ablations. We ablate Munin's key design choices and report a detailed summary in Appendix D. Overall, (i) increasing the trajectory deviation budget η_{traj} yields a smooth speed–fidelity knob (more reuse equals higher speedup with minor performance loss); (ii) split-conformal calibration is necessary

to reliably achieve a target violation probability $\hat{p}_{\text{viol}} \approx \alpha$, while heuristic rules miscalibrate; (iii) sensitivity-aware budgeting via sampler coefficients $\{L_t\}$ improves the speed–safety trade-off by allocating recomputation to high-impact timesteps; (iv) shallow stem probes provide the best net speedups (good predictiveness at low overhead); (v) forbidding reuse in a small prefix/suffix further reduces collisions/violations with modest speed loss; and (vi) calibration is sample-efficient, with risk and speed stabilizing after a modest number of rollouts.

VI. Conclusion

This work introduces Muninn, a training-free caching wrapper that accelerates trajectory diffusion models without modifying model weights or the sampler. Muninn leverages two signals that are present in trajectory diffusion planners: (i) a cheap probe of how the model's internal representation evolves across denoising steps, and (ii) analytic sampler sensitivity coefficients that quantify how denoiser perturbations propagate into the final trajectory. By calibrating probe dynamics against potential reuse error, Muninn produces step-wise, distribution-free error envelopes that can be converted into a trajectory-level deviation budget. At run time, Muninn spends this budget to decide when cached denoiser outputs can be safely reused. Across simulated benchmarks and hardware experiments, Muninn substantially reduces denoiser evaluations and wall-clock latency while preserving task performance and safety metrics. A promising direction for future work is to replace the current fixed user-specified deviation budget with a context-aware budget that automatically tightens in cluttered or contact-rich regimes and relaxes in open-space or low-risk regimes.

Acknowledgements

This work was supported in part by NSF grants 2118329, IIS-2024733 and IIS-2331908, the Office of Naval Research grants N00014-23-1-2505, N00014-25-1-2519, N00014-23-1-2789, the U.S. Department of Defense grant 78170-RT-REP, and by the Army Research Laboratories under contract W911NF1920243. This is contribution #2135 from the Institute of Environment at Florida International University.

References

[1] Shubham Agarwal, Subrata Mitra, Sarthak Chakraborty, Srikrishna Karanam, Koyel Mukherjee, and Shiv Kumar Saini. Approximate caching for efficiently serving Text-to-Image diffusion models. In *USENIX Symposium on Networked Systems Design and Implementation*, pages 1173–1189, 2024.

[2] Anurag Ajay, Yilun Du, Abhi Gupta, Joshua B. Tenenbaum, Tommi Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision-making? *arXiv preprint arXiv:2211.15657*, 2022.

[3] Kevin Black, Michael Janner, Yilun Du, Ilya Kostrikov, and Sergey Levine. Training diffusion models with reinforcement learning. *arXiv preprint arXiv:2305.13301*, 2023.

[4] Daniel Bolya and Judy Hoffman. Token merging for fast stable diffusion. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4599–4603, 2023.

[5] Jiazi Bu, Pengyang Ling, Yujie Zhou, Yibin Wang, Yuhang Zang, Tong Wu, Dahua Lin, and Jiaqi Wang. Dicache: Let diffusion model determine its own cache. *arXiv preprint arXiv:2508.17356*, 2025.

[6] Joao Carvalho, An T Le, Mark Baierl, Dorothea Koert, and Jan Peters. Motion planning diffusion: Learning and planning of robot motions with diffusion models. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1916–1923, 2023.

[7] Guanjie Chen, Xinyu Zhao, Yucheng Zhou, Tianlong Chen, and Cheng Yu. Accelerating vision diffusion transformers with skip branches. *arXiv preprint arXiv:2411.17616*, 2024.

[8] Yuzhu Chen, Fengxiang He, Shi Fu, Xinmei Tian, and Dacheng Tao. Adaptive time-stepping schedules for diffusion models. In *Conference on Uncertainty in Artificial Intelligence*, 2024.

[9] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

[10] Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *IEEE International Conference on Robotics and Automation*, pages 4693–4700, 2018.

[11] Justin Fu, Aviral Kumar, Ofir Nachum, G. Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[12] Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

[13] Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. Social GAN: Socially acceptable trajectories with generative adversarial networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2255–2264, 2018.

[14] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

[15] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Neural Information Processing Systems*, volume 33, pages 6840–6851, 2020.

[16] Jonathan Ho, Chitwan Saharia, William Chan, David J. Fleet, Mohammad Norouzi, and Tim Salimans. Cascaded diffusion models for high fidelity image generation. *Journal of Machine Learning Research*, 23(47):1–33, 2022.

[17] Renming Huang, Yunqiang Pei, Guoqing Wang, Yangming Zhang, Yang Yang, Peng Wang, and Hengtao Shen. Diffusion models as optimizers for efficient planning in offline RL. In *European Conference on Computer Vision*, pages 1–17, 2024.

[18] Siyuan Huang, Zan Wang, Puhao Li, Baoxiong Jia, Tengyu Liu, Yixin Zhu, Wei Liang, and Song-Chun Zhu. Diffusion-based generation, optimization, and planning in 3D scenes. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16750–16761, 2023.

[19] Boris Ivanovic and Marco Pavone. The Trajectron: Probabilistic multi-agent trajectory modeling with dynamic spatiotemporal graphs. In *IEEE/CVF International Conference on Computer Vision*, pages 2375–2384, 2019.

[20] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5:3019–3026, 2019.

[21] Michael Janner, Yilun Du, Joshua B Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. *arXiv preprint arXiv:2205.09991*, 2022.

[22] Hyeonseong Jeon, Cheolhong Min, and Jaesik Park. Tree-guided diffusion planner. *arXiv preprint arXiv:2508.21800*, 2025.

[23] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *IEEE International Conference on Robotics and Automation*, pages 4569–4574, 2011.

[24] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. In *Neural Information Processing Systems*, volume 35, pages 26565–26577, 2022.

[25] Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[26] Namhoon Lee, Wongun Choi, Paul Vernaza, Christopher B Choy, Philip HS Torr, and Manmohan Chandraker. Desire: Distant future prediction in dynamic scenes with interacting agents. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 336–345, 2017.

[27] Lijiang Li, Huixia Li, Xiawu Zheng, Jie Wu, Xuefeng Xiao, Rui Wang, Min Zheng, Xin Pan, Fei Chao, and Rongrong Ji. Autodiffusion: Training-free optimization of time steps and architectures for automated diffusion model acceleration. In *IEEE/CVF International Conference on Computer Vision*, pages 7105–7114, 2023.

[28] Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *International Conference on Informatics in Control, Automation and Robotics*, pages 222–229, 2004.

[29] Zhixuan Liang, Yao Mu, Mingyu Ding, Fei Ni, Masayoshi Tomizuka, and Ping Luo. AdaptDiffuser: Diffusion models as adaptive self-evolving planners. *arXiv preprint arXiv:2302.01877*, 2023.

[30] Weijie Liu, Peng Zhou, Zhiruo Wang, Zhe Zhao, Haotang Deng, and Qi Ju. FastBERT: A self-distilling BERT with adaptive inference time. In *Annual Meeting of the Association for Computational Linguistics*, pages 6035–6044, 2020.

[31] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. DPM-Solver: A fast ODE solver for diffusion probabilistic model sampling in around 10 steps. In *Neural Information Processing Systems*, volume 35, pages 5775–5787, 2022.

[32] Haofei Lu, Dongqi Han, Yifei Shen, and Dongsheng Li. What makes a good diffusion planner for decision making? *arXiv preprint arXiv:2503.00535*, 2025.

[33] Kairong Luo, Caiwei Xiao, Zhiao Huang, Zhan Ling, Yunhao Fang, and Hao Su. Dreamfuser: Value-guided diffusion policy for offline reinforcement learning.

[34] Yunhao Luo, Chen Sun, Joshua B. Tenenbaum, and Yilun Du. Potential-based diffusion motion planning. *arXiv preprint arXiv:2407.06169*, 2024.

[35] Yunhao Luo, Utkarsh A. Mishra, Yilun Du, and Danfei Xu. Generative trajectory stitching through diffusion composition. *arXiv preprint arXiv:2503.05153*, 2025.

[36] Zhengyao Lv, Chenyang Si, Junhao Song, Zhenyu Yang, Yu Qiao, Ziwei Liu, and Kwan-Yee K Wong. Fastercache: Training-free video diffusion model acceleration with high quality. *arXiv preprint arXiv:2410.19355*, 2024.

[37] Zhaoyang Lyu, Xudong Xu, Ceyuan Yang, Dahua Lin, and Bo Dai. Accelerating diffusion models via early stop of the diffusion process. *arXiv preprint arXiv:2205.12524*, 2022.

[38] Xinyin Ma, Gongfan Fang, Michael Bi Mi, and Xinchao Wang. Learning-to-cache: Accelerating diffusion transformer via layer caching. In *Neural Information Processing Systems*, volume 37, pages 133282–133304, 2024.

[39] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Deepcache: Accelerating diffusion models for free. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15762–15772, 2024.

[40] Xinyin Ma, Runpeng Yu, Gongfan Fang, and Xinchao Wang. DKV-Cache: The cache for diffusion language models. *arXiv preprint arXiv:2505.15781*, 2025.

[41] David Q Mayne, James B Rawlings, Christopher V Rao, and Pierre OM Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000.

[42] Chenlin Meng, Robin Rombach, Ruiqi Gao, Diederik Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On distillation of guided diffusion models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14297–14306, 2023.

[43] Taehong Moon, Moonseok Choi, EungGu Yun, Jongmin Yoon, Gayoung Lee, and Juho Lee. Early exiting for accelerated inference in diffusion models. In *ICML Workshop on Structured Probabilistic Inference & Generative Modeling*, 2023.

[44] Taehong Moon, Moonseok Choi, EungGu Yun, Jongmin Yoon, Gayoung Lee, Jaewoong Cho, and Juho Lee. A simple early exiting framework for accelerated sampling in diffusion models. *arXiv preprint arXiv:2408.05927*, 2024.

[45] Alexander Quinn Nichol and Prafulla Dhariwal. Improved

denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pages 8162–8171, 2021.

[46] Chaoyi Pan, Zeji Yi, Guanya Shi, and Guannan Qu. Model-based diffusion for trajectory optimization. In *Neural Information Processing Systems*, volume 37, pages 57914–57943, 2024.

[47] Aaditya Prasad, Kevin Lin, Jimmy Wu, Linqi Zhou, and Jeannette Bohg. Consistency policy: Accelerated visuomotor policies via consistency distillation. *arXiv preprint arXiv:2405.07503*, 2024.

[48] Gokul Puthumanaillam, Aditya Penumarti, Manav Vora, Paulo Padrao, Jose Fuentes, Leonardo Bobadilla, Jane Shin, and Melkior Ornik. Belief-conditioned one-step diffusion: Real-time trajectory planning with just-enough sensing. In *Conference on Robot Learning*, pages 68–92, 2025.

[49] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.

[50] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics*, pages 627–635, 2011.

[51] Kallol Saha, Vishal Mandadi, Jayaram Reddy, Ajit Srikanth, Aditya Agarwal, Bipasha Sen, Arun Singh, and Madhava Krishna. EDMP: Ensemble-of-costs-guided diffusion for motion planning. In *IEEE International Conference on Robotics and Automation*, pages 10351–10358, 2024.

[52] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022.

[53] Christoph Schöller and Alois Knoll. Flomo: Tractable motion prediction with normalizing flows. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 7977–7984, 2021.

[54] John Schulman, Jonathan Ho, Alex X Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: Science and Systems*, 2013.

[55] Mingyo Seo, Yoonyoung Cho, Yoonchang Sung, Peter Stone, Yuke Zhu, and Beomjoon Kim. Presto: Fast motion planning using diffusion models based on key-configuration environment representation. In *IEEE International Conference on Robotics and Automation*, pages 10861–10867, 2025.

[56] Yorai Shaoul, Itamar Mishani, Shivam Vats, Jiaoyang Li, and Maxim Likhachev. Multi-robot motion planning with diffusion models. *arXiv preprint arXiv:2410.03072*, 2024.

[57] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[58] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

[59] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. 2023.

[60] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *International Conference on Pattern Recognition*, pages 2464–2469, 2016.

[61] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. SkipNet: Learning dynamic routing in convolutional networks. In *European Conference on Computer Vision*, pages 409–424, 2018.

[62] Zhendong Wang, Jonathan J. Hunt, and Mingyuan Zhou. Diffusion policies as an expressive policy class for offline reinforcement learning. *arXiv preprint arXiv:2208.06193*, 2022.

[63] Zhendong Wang, Zhaoshuo Li, Ajay Mandlekar, Zhenjia Xu, Jiaojiao Fan, Yashraj S. Narang, Linxi Fan, Yuke Zhu, Yogesh Balaji, Mingyuan Zhou, Ming-Yu Liu, and Yuan Zeng. One-step diffusion policy: Fast visuomotor policies via diffusion distillation. *arXiv preprint arXiv:2410.21257*, 2024.

[64] Grady Williams, Andrew Aldrich, and Evangelos Theodorou. Model predictive path integral control: From theory to parallel computation. *Journal of Guidance, Control, and Dynamics*, 40(2):344–357, 2017.

[65] Rosa Wolf, Yitian Shi, Sheng Liu, and Rania Rayyes. Diffusion models for robotic manipulation: A survey. *arXiv preprint arXiv:2504.08438*, 2025.

[66] Yiming Wu, Huan Wang, Zhenghao Chen, Jianxin Pang, and Dong Xu. On-device diffusion transformer policy for efficient robot manipulation. *arXiv preprint arXiv:2508.00697*, 2025.

[67] Zhou Xian, Nikolaos Gkanatsios, Theophile Gervet, Tsung-Wei Ke, and Katerina Fragkiadaki. Chaineddiffuser: Unifying trajectory diffusion and keypose prediction for robotic manipulation. In *Conference on Robot Learning*, 2023.

[68] Sirui Xie, Zhisheng Xiao, Diederik Kingma, Tingbo Hou, Ying Nian Wu, Kevin P Murphy, Tim Salimans, Ben Poole, and Ruiqi Gao. EM distillation for one-step diffusion models. In *Neural Information Processing Systems*, volume 37, pages 45073–45104, 2024.

[69] Tianwei Yin, Michaël Gharbi, Richard Zhang, Eli Shechtman, Fredo Durand, William T Freeman, and Taesung Park. One-step diffusion with distribution matching distillation. *arXiv preprint arXiv:2311.18828*, 2023.

[70] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan C. Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*, 2019.

[71] Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3d diffusion policy:

Generalizable visuomotor policy learning via simple 3d representations. In *Robotics: Science and Systems (RSS)*, 2024.

[72] Zheng Zhan, Yushu Wu, Yifan Gong, Zichong Meng, Zhenglun Kong, Changdi Yang, Geng Yuan, Pu Zhao, Wei Niu, and Yanzhi Wang. Fast and memory-efficient video diffusion using streamlined inference. In *Neural Information Processing Systems*, volume 37, pages 13660–13684, 2024.

[73] Hui Zhang, Zuxuan Wu, Zhen Xing, Jie Shao, and Yu-Gang Jiang. AdaDiff: Adaptive step selection for fast diffusion models. In *AAAI Conference on Artificial Intelligence*, volume 39, pages 9914–9922, 2025.

[74] Zhengbang Zhu, Hanye Zhao, Haoran He, Yichao Zhong, Shenyu Zhang, Yong Yu, and Weinan Zhang. Diffusion models for reinforcement learning: A survey. *arXiv preprint arXiv:2311.01223*, 2023.

[75] Zhengbang Zhu, Minghuan Liu, Liyuan Mao, Bingyi Kang, Minkai Xu, Yong Yu, Stefano Ermon, and Weinan Zhang. MaDiff: Offline multi-agent learning with diffusion models. In *Neural Information Processing Systems*, volume 37, pages 4177–4206, 2024.

[76] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa. CHOMP: Covariant Hamiltonian optimization for motion planning. *International Journal of Robotics Research*, 32(9-10): 1164–1193, 2013.