arXiv:2603.01023v1 [cs.RO] 1 Mar 2026

面向闭环自动驾驶中基于扩散的运动规划的开源模块化基准

Li Yun[1], Simon Thompson[2], Yidu Zhang[1],Ehsan Javanmardi[1],和 Manabu Tsukada[1]

摘要— **基于扩散的运动规划器已在 nuPlan 等基准测试中取得最先进的成果，但在闭环量产自动驾驶软件栈中的评估仍鲜有探索。现有评估忽略了 ROS 2 通信延迟和实时调度约束，而单体式 ONNX 部署则会在导出时固定所有求解器参数。我们提出一种开源模块化基准，以弥补这两方面的不足：利用 ONNX GraphSurgeon，将一个包含 18,398 个节点的单体式扩散规划器拆分为三个可独立执行的模块，并以原生 C 重新实现 DPM-Solver++去噪循环。该系统作为 ROS2 节点集成到 Autoware 中。Autoware 是一款已部署于全球实车的开源自动驾驶软件栈；此系统无需重新编译模型即可在运行时配置求解器参数，并可逐步观测去噪过程，从而打破单体式部署的黑箱。与 CARLA 等独立仿真器中的评估不同，我们的基准运行于生产级软件栈中，并通过 AWSIM 闭环仿真进行验证。通过系统比较 DPM-Solver++ （一阶和二阶）与 DDIM 在六种步数配置（$N \in \{3, 5, 7, 10, 15, 20\}$）下的表现，我们发现编码器缓存可降低 $3.2\times$延迟，且在 $N=3$ 下，二阶求解相较于一阶可将 FDE 降低 41%。完整代码库将以开源形式发布，为从仿真基准迈向实车部署提供直接路径。**

I. 引言

扩散模型已成为自动驾驶（AD）运动规划的一种强大范式 [1]–[4]。基于从学习得到的数据分布中进行迭代去噪，将轨迹生成表述为这一过程，基于扩散模型的规划器能够联合建模多智能体的未来状态，并生成时间一致的规划。近期研究，如 Diffusion Planner [1]、DiffusionDrive [2]，和 Latent Planner（LAP） [4]，在 nuPlan基准 [5], [6]上报告了最先进的结果。然而，将这些基准测试结果转化为可部署系统仍然具有挑战性，原因是存在两个尚未得到充分解决的问题。

几乎所有现有的扩散规划器都在伪闭环设置下进行评估，这些设置通过基于 Python 的仿真器或 nuPlan 的反应式智能体 [5]回放录制场景。尽管这类设置有助于比较模型，却掩盖了生产级自动驾驶技术栈的延迟约束：ROS2 话题通信开销 [7], [8]，、传感器预处理流水线，以及共享车辆上的实时调度计算。因此，“基准测试性能”与“可部署于车辆的性能”之间的差距仍未得到衡量。如 §V-C所示，本文研究的单体式扩散规划器在 CPU 上每个规划周期需要超过 300 毫秒，远超 Autoware 级规划节点通常 100 毫秒的预算 [7]。

此外，当前的部署流水线会将整个多步去噪过程（包括噪声调度、求解器逻辑和迭代 DiT 模块）导出到一个单体 ONNX 图中。对于 Diffusion Planner [1], ，这会生成 18,398个节点，随后将其编译为 TensorRT 引擎。这种设计会在导出时固定所有求解器参数：步数（N=10）、求解器阶数和噪声调度都无法更改，除非重新导出并编译模型。这使研究人员无法探究一些基本问题，例如：在实时约束下，究竟需要多少去噪步骤？在闭环驾驶中，二阶求解相比一阶求解能带来显著收益吗？能否实现随时规划 *[9]* ，通过自适应地截断扩散步骤来完成？

这两个问题都源于一个设计选择：将迭代去噪循环嵌入计算图中。我们的核心思路是将循环移到外部，用可独立执行的模块取代单体模型，并由外部 C++ 求解器协调这些模块。具体而言，我们使用 ONNX GraphSurgeon [10]将单体图分解为三个模块：上下文编码器（3,417 个节点，18MB），每个规划周期执行一次并缓存结果；DiT核心（1,237 个节点，28MB），调用 N 次，其中 N可在运行时配置；以及转向指示器分类器（7个节点，5.6KB），用于事后信号预测。

DPM-Solver++ [11] 循环采用原生 C++ 重新实现，并支持在运行时配置步数（$N \in [1, 50]$）、求解器阶数和 VP 噪声调度 [12]参数。该系统作为即插即用的 ROS2 节点集成到 Autoware [13] 栈中，并在 AWSIM [14], 中完成验证。AWSIM 是一款开源仿真器，专为 Autoware 生态系统中的闭环评估而设计。模块化设计提供了可插拔的求解器接口：研究人员可以替换为其他去噪策略（DDIM [15], 、更高阶的 DPM-Solver 变体或新型调度器），无需修改神经网络权重或重新训练，即可立即在真实的闭环条件下进行评估。

我们的贡献如下。

首先，我们提出一个开源模块化基准，将单体式 ONNX 扩散规划器

[1]东京大学信息科学与技术研究生院，日本东京。{li-yun, zhangyidu7,ejavanmardi}@g.ecc.u-tokyo.ac.jp, tsukada@hongo.wide.ad.jp[2]TIER IV, Inc.，日本东京。simon.thompson@tier4.jp

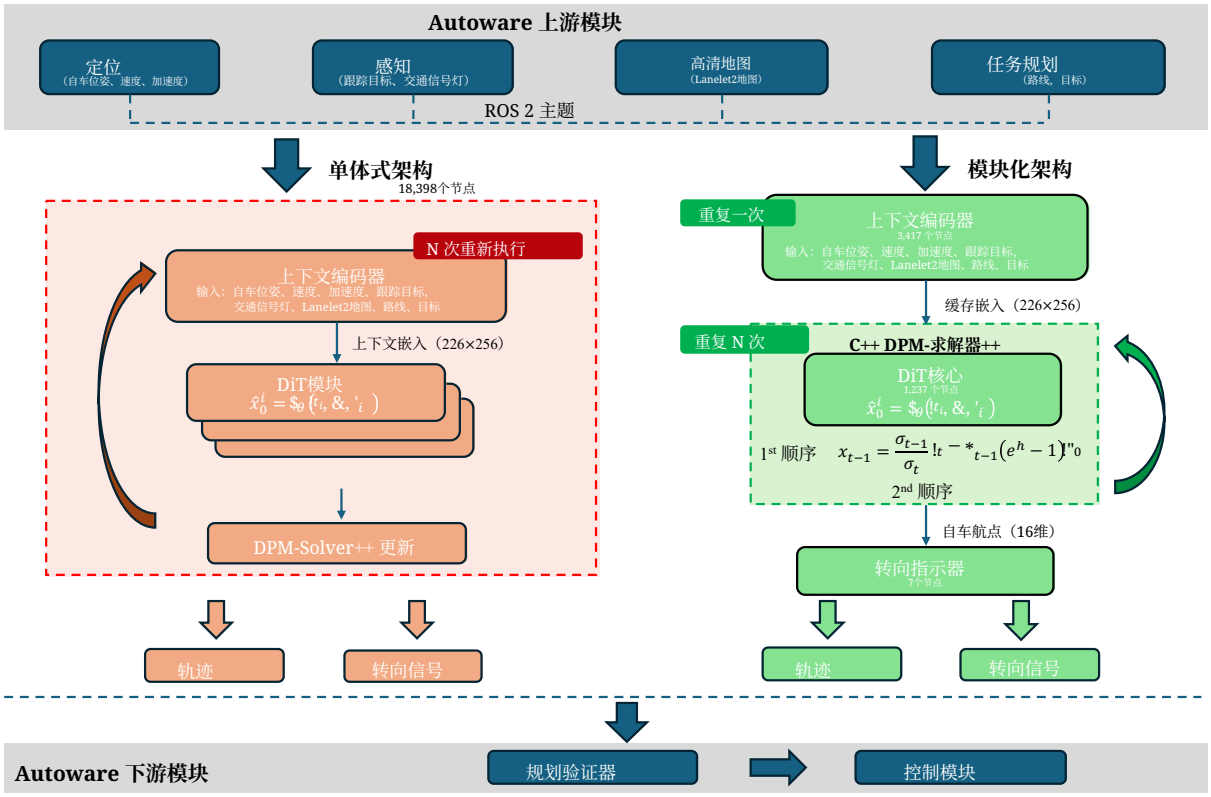

图 1. **(a)** 单体 ONNX 图 (18,398个节点) 编译为 TensorRT . **(b)**模块化分解：三个可独立执行的模块，具有 C++ 中的 DPM-Solver++ 循环文本编码器运行一次，并在所有去噪步骤中缓存。

通过 GraphSurgeon 拆分为可独立执行的模块，从而能够在 Autoware+ AWSIM 生产级自动驾驶技术栈中，在真实闭环条件下即插即用地评估求解器。

其次，我们提供了一个原生 C++ DPM-Solver++ 实现，其推理参数可在运行时配置。该实现表明，无需重新编译模型即可研究求解器行为，包括步数、阶数和噪声调度；现有的所有扩散规划器部署均不具备这一能力。

第三，通过在带有 $N \in \{3, 5, 7, 10, 15, 20\}$ 的 50 个 AWSIM帧上，对 DPM-Solver++ （一阶和二阶）与 DDIM 进行系统的求解器比较，我们表明，二阶校正可在 $N{=}3$ 时将 FDE 降低 41%，而恰当的时间步调度相比朴素的提前停止可将质量提升 $7\times$ 。

最后，我们验证了其与单体模型的数值等价性（最大误差 $< 10^{-5}$），展示了编码器缓存的效果（计算量减少 90.9%，延迟提升 3.2×），并确认其在 AWSIM 中成功实现城市驾驶。完整代码库将以开源形式发布。

II. 相关工作

A. 用于运动规划的扩散模型

扩散模型 [12]，[16]已越来越多地应用于自动驾驶领域。Diffuser [17] 开创了基于扩散的机器人轨迹规划；Diffusion Policy [18] 则展示了通过动作扩散实现视觉运动控制。在自动驾驶领域，Diffusion Planner [1] 采用 DiT 架构 [19]和方差保持（VP）噪声调度方案，对自车和邻车轨迹进行联合建模，并在 nuPlan基准 [5]上取得了最先进的表现。DiffusionDrive [2]提出了带锚定初始化的截断扩散，将步数减少至 2。BridgeDrive [3]将规划表述为扩散桥；LAP [4] 利用细粒度潜在特征加速推理；Diffusion-ES [20]将扩散与无梯度优化相结合；DriveLaW [21] 则将规划与视频生成统一起来。尽管算法取得了这些进展，所有 这些方法都是通过数据集回放或基于 Python 的仿真器进行评估，而非在满足实时约束的生产级自动驾驶技术栈中进行评估。

B. 生产技术栈中的闭环评估

nuPlan基准 [5]通过使用反应式智能体重放场景，提供“闭环”评估。然而，这只是伪闭环：规划器在 Python 进程中运行，不受生产系统中的中间件、通信和

调度约束影响。在现实世界的自动驾驶部署中，ROS2 是标准的中间件框架，为感知、规划和控制模块之间提供发布-订阅通信基础设施。Dauner 等人 [6]指出，即使是简单的基于规则的规划器，也能在 nuPlan 上取得有竞争力的分数，这让人质疑基准排名能否反映现实世界的性能。Betz 等人 [7] 进一步证明，基于 ROS2 的自动驾驶软件栈会引入显著的通信延迟，并沿整个处理流水线逐级传递，从根本上改变运行条件。我们的工作通过在 Autoware [13], 这一基于 ROS2 的全开源自动驾驶软件栈中进行基准测试，弥合了这一差距；全球各地的组织已将其用于实车部署。与百度 Apollo 等专有平台不同，Autoware 的开源特性支持完全可复现的研究，并能直接从仿真过渡到车载部署。

C. 模型部署与分区

在车辆平台上部署深度神经网络模型，通常需要先导出为 ONNX，再进行硬件专用编译 [23], [24]。Guo 等人 [25] 提出了 EASTER，该方法能够学习 Transformer 在边缘设备间的最优切分点。Nair 等人 [26] 提出 SecureFrameNet，用于在安全部署场景中对 ONNX 模型进行分区。Jajal 等人 [27] 分析了 ONNX 转换器互操作性故障，并发现深度学习框架中存在系统性问题。然而，这些方法面向通用模型，并未利用多步扩散模型的迭代结构；在这种结构中，同一去噪子网络会在求解器循环中被反复调用。

D. 扩散推理加速

另一类相关研究则着眼于加速扩散采样过程本身。DPM-Solver++ [11]是一种用于扩散模型的高阶常微分方程求解器，可将所需去噪步数从 DDPM [16] 中的数百步减少至 10–20 步，同时保持样本质量。DDIM [15] 通过隐式非马尔可夫过程实现确定性采样。截断扩散 [2] 等步数缩减方法可以进一步减少步数，但需要特定模型训练。近来，流匹配和一致性模型实现了单步或两步生成，但它们在生产级自动驾驶规划中的应用仍然有限。我们的工作与此正交且互为补充。我们不提出新的求解器，而是提供一个开源平台，可接入任意求解器，并在闭环驾驶条件下对其进行评估。

III. 系统架构

A. Autoware 技术栈中的 *Diffusion Planner*

图 1 概述了系统架构。在 Autoware 技术栈中，Diffusion Planner 作为规划模块运行，接收感知输出，包括跟踪目标、高清矢量地图、路线和交通信号灯，并生成由下游控制模块执行的轨迹。Diffusion Planner [1] 本身由三个功能组件组成：

上下文编码器。 编码器处理场景上下文 $\mathcal{I} = \{s_{\text{ego}}, s_{\text{nbr}}, s_{\text{lane}}, s_{\text{route}}, s_{\text{tl}}, s_{\text{goal}}\}$，其中包括自车历史、邻车轨迹、车道几何、路线信息、交通信号灯和目标位姿。这些异构输入通过混合器注意力架构融合为上下文嵌入 $\mathbf{c} \in \mathbb{R}^{L\times d}$ (L=226, d=256)。

扩散变换器（DiT）。DiT 主干网络以带噪声的联合轨迹 $\mathbf{x}_t \in \mathbb{R}^{A\times T\times 4}$ (A=33个智能体， T=81 个时间步，状态（$x, y,$ $\cos\theta, \sin\theta$）)、上下文嵌入 $\mathbf{c}$ 和连续时间步 $t \in [0,1]$ 为输入，并输出去噪后的轨迹估计 $\hat{\mathbf{x}}_0 \in \mathbb{R}^{A\times(T-1)\times 4}$。

转向指示器预测器。 线性分类器将上下文嵌入$\mathbf{c}$ 和采样得到的自车航点映射为 4 类转向指示器预测。

B. VP 噪声调度与 *DPM-Solver++*

扩散模型 [12], [16] 通过学习逆转添加噪声的前向过程来生成数据。给定一条干净轨迹 $\mathbf{x}_0$, ，前向过程会在连续时间 $t \in [0,1]$ 生成其含噪版本$\mathbf{x}_t = \alpha(t)\mathbf{x}_0 + \sigma(t)\boldsymbol{\epsilon}$ ，其中$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$。VP 噪声调度 [12] 将信号和噪声幅度参数化为：

$$\log\alpha(t) = -\tfrac{1}{4}t^2(\beta_1 - \beta_0) - \tfrac{1}{2}t\beta_0, \tag{1}$$

其中 $\sigma(t) = \sqrt{1-\alpha(t)^2}$。对数信噪比$\lambda(t) = \log[\alpha(t)/\sigma(t)]$ 从干净数据（$t\approx 0$）到近似纯噪声（$t\approx 1$）单调递减。Diffusion Planner 使用 $\beta_0 = 0.1$ 和 $\beta_1 = 20.0$。

在推理阶段，DiT 根据含噪输入 $\mathbf{x}_t$ 和上下文 $\mathbf{c}$预测干净轨迹 $\hat{\mathbf{x}}_0$，求解器则从 t=1 朝 t=0迭代去噪。DPM-Solver++ [11] 是专用于该反向过程的高阶常微分方程求解器。从时间步 t_s 到 t的一阶更新为：

$$\mathbf{x}_t = \frac{\sigma_t}{\sigma_s}\mathbf{x}_s + \alpha_t(1 - e^{-h})\hat{\mathbf{x}}_0, \tag{2}$$

其中， $h = \lambda_t - \lambda_s$ 是两个时间步之间的 logSNR 差值。当存在先前的预测时，二阶更新会通过引入有限差分校正来改进估计：

$$\mathbf{x}_t = \frac{\sigma_t}{\sigma_s}\mathbf{x}_s + \alpha_t(1 - e^{-h})\hat{\mathbf{x}}_0 + \frac{1}{2}\alpha_t(1 - e^{-h})D_1, \tag{3}$$

其中， $D_1 = (\hat{\mathbf{x}}_0^{(i)} - \hat{\mathbf{x}}_0^{(i-1)})/r$ 是带有 $r = h_{i-1}/h_i$ 的有限差分导数。该修正利用连续预测来更好地近似去噪轨迹，在神经网络评估次数相同的情况下实现更高的准确性。

C. 单体式部署

在最初的 Autoware 集成中，整个扩散过程（包括 11 个 DiT 步骤、VP 调度和 DPM-Solver++逻辑）在导出时被展开为单个 ONNX 图。这样生成的模型包含**18,398个节点**，而单个 DiT 步骤仅包含 1,237 个节点，并被编译为 ∼68MB 的 TensorRT 引擎。虽然这实现了激进的内核融合，但也冻结了所有求解器参数，并迫使上下文编码器在每个去噪步骤中重新执行。每个规划周期的总推理成本为：

$$T_{\text{mono}} = (N{+}1) \cdot T_{\text{enc}} + (N{+}1) \cdot T_{\text{dit}} + N \cdot T_{\text{sol}}, \quad (4)$$

其中 T_{enc}、 T_{dit}和 T_{sol} 分别表示上下文编码器、单次 DiT 步骤和求解器更新的延迟。由于 $T_{\text{enc}} \gg T_{\text{dit}}$ （如§V-C所量化），冗余的 $(N{+}1)$ 编码器调用主导了总成本。图 1(a) 展示了这种单体式设计；我们提出的模块化替代方案详见§IV ，并如图 1(b) 所示。

IV. 方法

A. 通过 *ONNX GraphSurgeon* 进行图分解

如图 1(a) 所示，单体 ONNX 图包含一个重复子结构：相同的 DiT模块出现 11 次，具有共享权重，但时间步输入不同。我们利用这一结构将图分解为三个可独立执行的模块（图 1(b)）。

步骤 1：子图识别。 使用 ONNX GraphSurgeon [10], ，我们分析计算图以识别：(a) 上下文编码器子图，其输出馈入所有 DiT 模块，但不接收来自它们的反馈；(b) 单个 DiT模块，依据 11 个展开副本之间的权重共享加以识别；(c) 位于图输出端的转向指示器线性层。

步骤 2：子图提取。将每个子图提取为独立的 ONNX 模型，并明确指定其输入和输出。编码器的输出节点（上下文嵌入 $\mathbf{c}$）既作为编码器模型的输出，也作为 DiT 模型的输入。时间步在每个展开副本中原本是常量，提取后则成为 DiT 模型的动态输入。具体而言，我们使用 GraphSurgeon 在每个子图边界处创建新的输入变量节点，具体包括编码器最终的层归一化输出和 DiT 的输出投影；随后重新连接所有下游消费者节点，并调用自动图清理，剪除现已不可达的编码器节点和重复的 DiT 节点。转向指示器则根据提取出的线性层权重重建为一个最小化的 7 节点图。

步骤 3：验证。我们在相同输入上运行原始单体模型和分解后的处理流水线，以验证数值等价性，并确认逐元素最大误差低于 10^{-5}；参见 §V-A。

表I总结了分解过程。

与单体式 TensorRT 引擎不同，分解后的 ONNX 模块可以通过 ONNX Runtime [23] 在任何受支持的后端上执行，包括 CPU、CUDA 和其他加速器，无需重新编译。

表 I 单体式与模块化架构对比。

	单体式	模块化（我们的方法）
ONNX 节点	18,398	4,661†
模型大小	47 MB	46 MB
每周期编码器调用次数	$N{+}1$	1（已缓存）
运行时后端	TensorRT（GPU）	ONNX Runtime（CPU/GPU）
求解器配置	导出时冻结	运行时可配置

†编码器 (3,417) + DiT核心 (1,237) + 转向指示器 (7)。

算法 1 模块化扩散规划 使用 DPM-求解器++

要求： 场景输入 $\mathcal{I}$，步数 N，阶数 $p \in \{1, 2\}$

$\mathbf{c} \leftarrow$ 编码器($\mathcal{I}$) ▷ 运行一次并缓存

$\{t_0, \ldots, t_N\} \leftarrow$ VPSchedule(N, β_0, β_1)

$\mathbf{x}_{t_0} \leftarrow \mathbf{0}$ ▷ 零初始化（*temp.*$= 0$）

对于 $i = 0$ **到** $N-1$ **执行**

  $\hat{\mathbf{x}}_0^{(i)} \leftarrow \text{DiT}(\mathbf{x}^{t_i}, \mathbf{c}, t_i)$ ▷ 单步

  如果 $i = 0$ **或** $p = 1$ **则**

    $\mathbf{x}^{t_{i+1}} \leftarrow$公式（2） ▷ *1*阶

  else

    $\mathbf{x}_{t_{i+1}} \leftarrow$ 公式（3） ▷ *2*阶

  如果结束

  $\mathbf{x}_{t_{i+1}}$[自车 0] ← **秒**$_{\text{cur}}$ ▷ 约束

</style}结束循环

$\hat{\mathbf{x}}_0 \leftarrow \text{DiT}(\mathbf{x}^{t_N}, \mathbf{c}, t_N)$ ▷ 去噪至零

$\mathbf{y}_{\text{turn}} \leftarrow$TurnInd($\hat{\mathbf{x}}_0, \mathbf{c}$)

$\hat{\mathbf{x}}_0 \leftarrow$反归一化($\hat{\mathbf{x}}_0$)

返回 $\hat{\mathbf{x}}_0, \mathbf{y}_{\text{turn}}$

B. C++DPM-Solver++实现

将 DiT 核心提取为单步模型后，我们用原生 C++重新实现了 DPM-Solver++ 去噪循环。这一选择由 Autoware 生态系统的需求驱动：Autoware 节点以 C++实现，ONNX Runtime C++ API 可避免 Python 解释器开销和全局解释器锁带来的限制，而确定性内存管理对于满足生产级自动驾驶技术栈的实时调度要求至关重要。算法1展示了完整流程。

时间步调度采用 logSNR 均匀间隔 [11]：给定 N 个时间步，我们将 logSNR 范围 $[\lambda_{\min}, \lambda_{\max}]$ 均匀划分为 $N{+}1$ 个值，并对每个值求逆以获得连续时间步 $t_i \in (0, 1]$。这种方式会在高噪声区域集中分配时间步，因为去噪轨迹在该区域变化最快。每次求解器更新后，都会用观测到的自车和邻近智能体状态覆盖轨迹张量中当前时间步对应的切片，使预测锚定于物理现实，如算法1第9行所示。

模块化设计的一项主要优势是编码器缓存。由于上下文编码器输出 $\mathbf{c}$ 仅取决于场景上下文 $\mathcal{I}$ ，而不取决于去噪状态$\mathbf{x}_t$，因此可以只计算一次，并在所有 $N{+}1$ DiT 调用中重复使用，

将推理成本从 T_{mono} 公式4中的值降低至：

$$T_{\text{mod}} = T_{\text{enc}} + (N{+}1) \cdot T_{\text{dit}} + N \cdot T_{\text{sol}}. \tag{5}$$

节省的 $N \cdot T_{\text{enc}}$ 幅度可观，因为 $T_{\text{enc}} \gg T_{\text{dit}}$ (§V-C）。模型在归一化坐标系中运行；输出轨迹在发布为 ROS2 轨迹消息之前会反归一化至世界坐标，以便 Autoware 技术栈中的下游规划验证和车辆控制模块直接使用。

C. 与 *Autoware* 集成 *ROS 2*

模块化规划器作为原始 Autoware 扩散规划器节点的直接替代实现。ROS2 节点订阅里程计、跟踪目标、矢量地图、路线和交通信号话题，执行与原始节点相同的预处理，并发布轨迹和转向指示灯输出。所有求解器参数（N、p、 β_0、 β_1）均作为 ROS2 参数公开，因此可以通过 ros2 param set 在运行时重新配置，无需重启节点。

V. 实验

A. 实验设置

所有实验均在一台台式电脑上进行，配备 Intel Core i9-12900KS CPU、32GB 内存和英伟达 GeForce RTX 3090Ti GPU，并在运行 Ubuntu 22.04、ROS 2 Humble 和 Autoware 的 Docker 容器中开展。模块化规划器使用配备 CPU 执行提供程序的 ONNX Runtime 1.23；原始单体规划器 [28]使用 GPU 上的 TensorRT 和 68 MB 引擎文件。两种规划器均在同一台机器上进行基准测试；基于 CPU 的比较能够隔离编码器缓存的架构优势，不受硬件加速效果影响。在 AWSIM 中进行闭环评估 [14]，该开源模拟器与 Autoware 集成 [13]，评估路线位于西新宿地图中的一条城市驾驶路线，包含直路、交叉路口和交通参与者。除非另有说明，模块化规划器采用 $N = 10$ 步、二阶求解器（$p = 2$）、VP 调度，取值为 $\beta_0 = 0.1$， $\beta_1 = 20.0$，温度为 $= 0$，并使用零初始噪声进行确定性采样，与原始部署保持一致。图2 展示了评估环境：Autoware 技术栈的 RViz 可视化界面（左）和 AWSIM 三维仿真（右）。

为验证图分解能够保留模型语义，我们使用采集的 AWSIM 驾驶数据，对比原始单体模型 [28] 与模块化处理流水线的输出。对于每个输出张量 $\mathbf{y}$，我们测量逐元素最大绝对误差：

$$\epsilon = \max_i \left| y_i^{\text{mono}} - y_i^{\text{mod}} \right|. \tag{6}$$

对于上下文编码器等单个模块，以及
单个 DiT 步骤为 $\epsilon < 10^{-5}$。经过 10 个连续的求解器步骤，累积的浮点舍入误差使最终轨迹产生 $\epsilon < 10^{-4}$ 。由于规划器在归一化

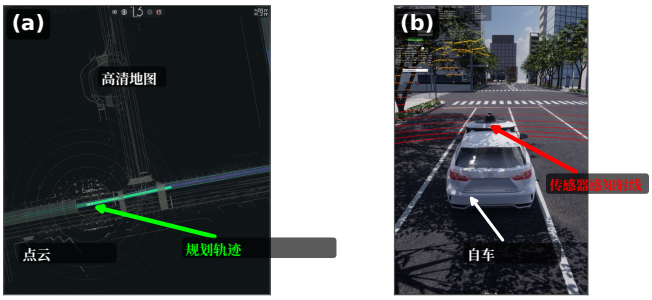

图2. 闭环评估环境。**(a)** RViz可视化展示了Autoware 技术栈中的高清地图、点云和规划轨迹。**(b)** 西新宿地图中城市交叉口处的AWSIM三维仿真，包含自车和传感器感知射线。

坐标系中运行，标准差为 $\sigma = 20$m，反归一化后 $x_{\text{phys}} = x_{\text{norm}} \cdot \sigma + \mu$，物理空间误差上限为 $\epsilon \cdot \sigma < 10^{-4} \times 20 = 2 \times 10^{-3}$m，即小于 2 mm，对规划而言可以忽略不计。

B. 去噪分析与求解器比较 *on*

模块化架构的一项关键能力是逐步可观测性：由于 C++ 求解器在每一步显式调用DiT核心，我们可以记录中间预测，而这些信息在单体式部署中仍是黑箱。图3（a）利用AWSIM驾驶帧的实际数据说明了这一点。去噪过程呈现三个不同阶段：粗略阶段（第1–2步），预测仍接近噪声先验且变化极小；细化阶段（第3–6步），两个轴向快速收敛，逐步变化最大；以及精细调整阶段（第7–10步），残余修正逐渐减小。横向位置以非单调方式细化，表明曲率调整滞后于纵向结构。图3（b）量化了收敛速度：纵向相对误差在第5步降至6.4%，在第7步降至3.1%以下，而横向通道因其非单调细化而收敛更慢。这一诊断能力与运行时可配置的 N、 p以及（β_0, β_1）相结合，使系统化的求解器分析无需模型重新编译即可进行。

由于细化阶段集中了大部分预测变化，减少步数主要会截短精细调整阶段。在 N=7，DiT 调用次数减少 27% 时，8 秒预测时域的平均航点偏差仅为 0.87 米，远在 10 Hz 规划器的重新规划裕度之内；该规划器每次只执行每条轨迹的前 1–2 秒。即使在 N=5，减少 45% 时，1.71 米的偏差对于延迟敏感场景也可能可以接受，这支持了随时规划的可行性 [9]，即步数可根据可用计算预算进行调整。

为进一步展示可插拔求解器接口，我们开展了一项离线基准测试：对于均匀采样的50个AWSIM帧中的每一帧，我们将捕获的输入张量通过DiT核心进行回放，并采用三种求解器配置

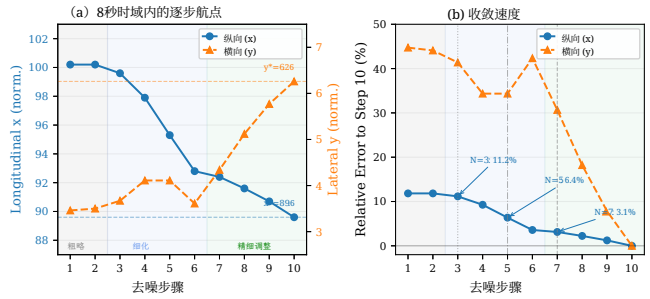

图 3. 8 秒预测时域路径点的去噪进程。 **(a)**每步预测位置的归一化坐标：纵向（蓝色）和横向（橙色）；可见三个阶段：粗略（步骤 1–2）、细化（3–6）和精细调整（7–10）。**(b)**相对于收敛后的第 10 步预测结果的相对误差。

（DPM-Solver++ 一阶（p=1）、二阶（p=2）以及 DDIM [15]），遍及 $N \in \{3, 5, 7, 10, 15, 20\}$。每种配置均使用各自针对给定 N优化的对数信噪比均匀时间步调度。图5 报告了最终位移误差（FDE）和平均位移误差（ADE），相对于 N=10， p=2 参考。DPM-Solver++ 一阶在所有配置下产生与DDIM完全相同的输出，从经验上证实了数据预测模型已知的理论等价性 [11]。二阶求解器带来了显著提升。在 N=3处，FDE相对一阶下降41%，从0.61 m降至0.36m，且该改进在所有 N上均保持一致，其中 p=2 以大约一半的步数达到与 p=1 相同的精度。恰当的时间步调度同样重要：与上文的早停分析相比，专用 N=3 调度得到的FDE为 0.36 m，而截断10步调度时为2.53 m，即 7×的提升。

由于模块化架构将求解器作为可配置组件开放出来，研究人员可以直接使用从生产级自动驾驶技术栈采集的驾驶数据，评估新颖的调度策略、步数自适应策略和替代求解器，而不必依赖在独立仿真器中重放数据集。

C. 系统性能与编码器缓存

前几节分析了算法收敛性，本节则评估自动驾驶中至关重要的系统级指标：推理延迟。

表 II 对推理延迟进行了分解。该延迟是在 CPU 上通过 ONNX Runtime 测得的，运行 100 次并报告中位数。上下文编码器占据了每个步骤的大部分开销，为T_{enc} = 27.5毫秒。在单体架构中，由于采用融合图结构，该编码器会在每个去噪步骤中重复执行，造成 90.9% 的计算浪费。我们的模块化架构只运行一次并缓存输出，因此每增加一个步骤的边际成本仅为T_{dit} = 2.3毫秒。请注意，表 II中的单体架构 328 毫秒，是根据相同的各组件延迟计算得出的等效 CPU 成本；实际的单体规划器运行在

表 II 推理延迟分解。

组件	Time	单声道调用	调制调用
上下文编码器	27.5 毫秒	N+1	1
DiT核心† （每步）	2.3 毫秒	N+1	N+1
C++ 求解器开销	0.01 毫秒/步	—	N
总计（N=10）		**328 毫秒**	**53 毫秒**

†N+1: N 求解器步数 + 1最终去噪至零（算法1，第10行）。

GPU 上，并通过 TensorRT 运行，因此该数值反映的是架构差异，而非硬件差异。

图4绘制了三种配置下推理延迟与 FDE 的关系；理想运行点位于左下角，此处延迟和预测误差均达到最小。图中的水平箭头展示了编码器缓存收益：在 N=3 下，使用相同的二阶求解器时，从单体式切换为模块化可将延迟从 119 毫秒降至 37 毫秒，提速 **3**。**2**× 倍，且 FDE 不变。垂直箭头展示了求解器阶数收益：在相同延迟下， N=3 的二阶求解精度可媲美 N=5的一阶求解，所需步数实际上减半。值得注意的是，所有模块化配置均在 100 毫秒规划预算内，而没有任何单体式配置能做到这一点；即使是最快的单体式方案，延迟也已超过 119 毫秒。这意味着规划器可以根据每个 100 毫秒 ROS 2 周期内剩余的时间，动态选择步数：计算资源充足时，使用 N=10 以获得最高精度；如果前一个模块发生超时，则退回到 N=3，仍能以可接受的质量按时完成。

我们在 AWSIM 闭环仿真器中验证了该系统：运行完整的 Autoware 技术栈，并将我们的节点作为原单体式规划器的即插即用替代品 [28]。在西新宿城市路线上的 600+ 个规划周期中，自车未发生碰撞并顺利完成路线，以稳定的 ∼10 Hz 输出速率运行，推理加预处理耗时为 53 ms。

VI. 讨论与结论

我们提出了一种开源模块化基准，将单体式 ONNX 扩散规划器拆解为可在 Autoware 量产级自动驾驶软件栈和 AWSIM 闭环仿真器中独立执行的模块。模块化设计解耦了架构优化与算法优化：仅采用编码器缓存即可降低 3.2× 的延迟，而升级至二阶求解器则可在 N=3 下将 FDE 降低 41%。这两项改进相互正交，且可叠加，同时保持数值等价性。

该分解仅依赖于以可缓存上下文为条件的重复子结构，这一模式已被近期的扩散规划器 [1]–[4] 和场景生成器 [29],采用，因此该方法易于迁移。据我们所知，目前还没有开源框架能在基于 ROS2 的自动驾驶软件栈中，在真正的闭环条件下提供可插拔求解器评估。

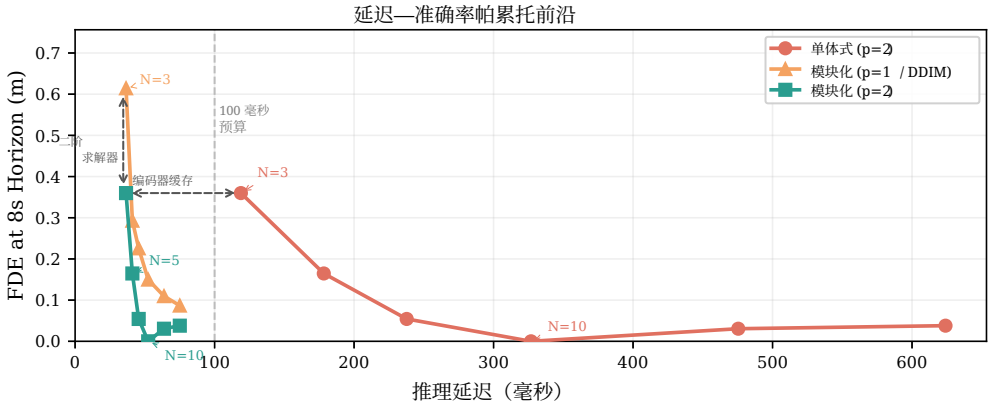

图4. 延迟–准确率帕累托前沿。水平差距：编码器缓存收益；垂直差距：求解器阶数收益。虚线：100ms 规划预算。

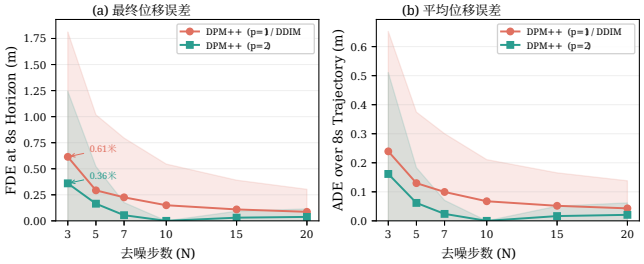

图5. 求解器比较。**(a)** FDE 和 **(b)** ADE 与 N=10, p=2 参考值的对比。阴影部分: $\pm 1\sigma$。

我们目前的评估仅涵盖 AWSIM 中的单一城市场景。更广泛的场景覆盖、系统性的 GPU 延迟分析，以及对极低步数下模块间传输开销的分析，将留待未来工作完成。完整代码库将以开源形式发布，以支持闭环扩散规划的可复现研究。

参考文献

[1] Y. Zheng, R. Liang, K. Zheng, J. Zheng, L. Mao, J. Li, W. Gu, R. Ai, S. E. Li, X. Zhan, and J. Liu, “面向自动驾驶的灵活引导式基于扩散的规划”, 载于 第十三届国际学习表征会议，2025年1月26日。 [2] B. Liao, S. Chen, H. Yin, B. Jiang, C. Wang, S. Yan, X. Zhang, X. Li, Y. Zhang, Q. Zhang, and X. Wang, “DiffusionDrive：用于端到端自动驾驶的截断扩散模型”，载于 *IEEE/CVF*计算机视觉与模式识别会议（*CVPR*）论文集。IEEE，2025年11月22日。 [在线]。可访问: https://openaccess.thecvf.com/content/CVPR2025/papers/ Liao DiffusionDriveTruncated Diffusion Model for End-to-EndAutonomous - Driving CVPR 2025paper.pdf [3] S. Liu, W. Chen, W. Li, Z. Wang, L. Yang, J. Huang, Y. Zhang, Z. Huang, Z. Cheng, and H. Yang, “BridgeDrive：面向自动驾驶闭环轨迹规划的扩散桥策略”，载于 第十四届国际学习表征会议，2026年10月8日。 [在线]。可访问: https://openreview.net/forum?id=dJKhjK4zpp [4] J. Zhang, W. Xia, Z. Zhou, Y. Gong, and J. Mei, “LAP：用于自动驾驶的快速潜在扩散规划器，采用细粒度特征蒸馏”，*arXiv [cs.RO]*, 2 2025年12月。 [在线]。可访问: http://arxiv.org/abs/2512.00470

[5] N . Karnchanachari, D. Geromichalos, K. S. Tan, N. Li, C. Eriksen, S. Yaghoubi, N. Mehdipour, G. Bernasconi, W. K. Fong, Y. Guo 和 H. Caesar, 《迈向基于学习的规划：面向真实世界自动驾驶的 nuPlan基准》，载于 *2024*年 *IEEE* 国际机器人与自动化会议 *(ICRA)*，第 56 卷。IEEE，2024年5月13日，第 629–636 页。 [在线]。可访问: http://dx.doi.org/10.1109/ICRA57147.2024.10610077

[6] D. Dauner、M. Hallgarten、A. Geiger 和 K. Chitta，“摒弃对基于学习的车辆运动规划的误解”，载于《机器人学习会议》。PMLR，2023年12月2日，第1268–1281页。 [在线]。获取地址：https://proceedings.mlr.press/v229/dauner23a.html[7]T. Betz、M. Schmeller、H. Teper 和 J. Betz，“我的软件有多快？ ROS 2 自动驾驶软件的延迟评估”，载于《2023年 IEEE 智能车辆研讨会》(IV)。IEEE，2023年6月4日，第1–6页。 [在线]。获取地址：http: //dx.doi.org/10 1109 55152 2023 10186585 . /IV . . T. Betz、H. Teper、D. [8] Ebner、M. Leitenstern、S. Sagmeister、M. Weinmann、J.-J. Chen 和 M. Lienkamp，“容器化 ROS 2 自动驾驶软件的端到端延迟优化”，IEEE Access，第13卷，第112 654–112 672页，2025年。 [在线]。获取地址：http://dx.doi.org/10 1109 2025 3582868 . /ACCESS. . M. Likhachev、G. [9] J. Gordon 和 S. Thrun，“ARA*：具有次优性可证明界限的任意时长 A*”，载于《神经信息处理系统进展》，第16卷，2003年12月9日，第767–774页。[在线]。获取地址：https://proceedings.neurips.cc/paper/2003/hash/ee8fe9093fbbb687bef15a38facc44d2-Abstract.html[10] 英伟达，“ONNX GraphSurgeon”，https://github.com/NVIDIA/TensorRT/tree/main/tools/onnx-graphsurgeon，2024年。[11] C. Lu、Y. Zhou、F. Bao、J. Chen、C. Li 和 J. Zhu，“DPM-solver++：用于引导采样扩散概率模型的快速求解器”，arXiv *[cs.LG]*, 2 ，2022年11月。 [在线]。获取地址：http://arxiv.org/abs/ 2211 01095 . Y. Song、J. [12] Sohl-Dickstein、D. P. Kingma、A. Kumar、S. Ermon 和 B. Poole，“通过随机微分方程进行基于分数的生成建模”，载于《国际学习表征会议》，2020年10月2日。 [在线]。获取地址：https: //openreview.net/forum?id=PxTIG12RRHS&utm campaign=NLP% 20News&utm medium=email&utm source=Revue% 20newsletter[13] Autoware基金会，“Autoware：介绍开源端到端自动驾驶”，Autoware基金会，技术报告，2025年，技术报告。获取地址：https://autoware.org/wp-content/ uploads/2025/10/Autoware-E2E-Report Release.pdf。[14] TIER IV, Inc.，“AWSIM：面向 Autoware 的开源自动驾驶仿真器”，https://github.com/tier4/AWSIM，2024年。[15] J. Song、C. Meng 和 S. Ermon，“去噪扩散隐式模型”，载于《国际学习表征会议》，2021年10月2日。 [在线]。获取地址：https://openreview.net/forum?id=St1giarCHLP [16]J. Ho、A. Jain 和 P. Abbeel，“去噪扩散

概率模型”，载于《神经信息处理系统进展》，第33卷，2020年，第6840–6851页。[在线]。获取地址：https://proceedings.neurips.cc/paper/2020/hash/4c5bcfec8584af0d967f1ab10179ca4b-Abstract.html[17]M. Janner、Y. Du、J. Tenenbaum 和 S. Levine，“基于扩散的灵活行为合成规划”，载于《国际机器学习会议》。PMLR，2022年6月28日，第9902–9915页。 [在线]。获取地址：https://proceedings.mlr.press/v162/ janner22a.html [18] C. Chi、S. Feng、Y. Du、Z. Xu、E. Cousineau、B. Burchfiel 和 S. Song，“Diffusion policy：通过动作扩散实现视觉运动策略学习”，载于《机器人：科学与系统》第十九届会议。机器人：科学与系统基金会，2023年7月10日。 [在线]。获取地址：http://dx.doi.org/10 15607 2023 026 . /rss. .xix. W. Peebles 和 S. Xie，“利[19]用 Transformer 实现可扩展的扩散模型”，载于 IEEE/CVF 计算机视觉国际会议论文集，2023年，第4195–4205页。 [在线]。获取地址：http: //openaccess.thecvf.com/content/ICCV2023/html/Peebles Scalable Diffusion Models with Transformers ICCV 2023 paper.html [20] B. Yang、H. Su、N. Gkanatsios、T.-W. Ke、A. Jain、J. Schneider 和 K. Fragkiadaki，“Diffusion-ES：用于自动驾驶和零样本指令跟随的无梯度扩散规划”，arXiv /cs.LG /, 9 ，2024年2月。 [在线]。获取地址：http://arxiv.org/abs/2402. 06559[21] T. Xia、Y. Li、L. Zhou、J. Yao、K. Xiong、H. Sun、B. Wang、K. Ma、G. Chen、H. Ye、W. Liu 和 X. Wang，“DriveLaW：在潜在驾驶世界中统一规划与视频生成”，arXiv [ ], cs.CV，12 月。在线。获取地址：http://arxiv.org/abs/。[22] 百度，“Apollo：开源自动驾驶平台”，https: //github.com/ApolloAuto/apollo，2024年。[23] 微软，“ONNX Runtime：跨平台、高性能的机器学习推理与训练加速器”，https://onnxruntime.ai/，2019年。[24] 英伟达，“NVIDIA TensorRT”，https://developer.nvidia.com/tensorrt，2024年。[25]X. Guo、Q. Jiang、Y. Shen、A. D. Pimentel 和 T. Stefanov，“EASTER：学习在边缘端稳健地拆分Transformer”，IEEE《计算机辅助设计与集成电路系统汇刊》，第43卷，第11期，第3626–3637页，2024年11月。 [在线]。获取地址：http: //dx.doi.org/10 1109 2024 3438995 . /TCAD. . R. C. Nair、M. Rao 和[26]Muralidhara，“SecureFrameNet：一种功能丰富的计算安全框架，用于在边缘受保护系统上部署神经网络模型”，载于《第三届人工智能与机器学习系统国际会议》。美国纽约：ACM，2023年10月25日，第1–13页。 [在线]。获取地址：http://dx.doi.org/10 1145 3639856 3639864 . / . P. [27]Jajal、W. Jiang、A. Tewari、E. Kocinare、J. Woo、A. Sarraf、Y.-H. Lu、G. K. Thiruvathukal 和 J. C. Davis，“深度学习中的互操作性：对 ONNX 模型转换器的用户调查与故障分析”，载于第33届 ACM SIGSOFT 软件测试与分析国际研讨会论文集，第18卷。美国纽约：ACM，2024年9月11日，第1466–1478页。 [在线]。获取地址：http://dx.doi.org/10 1145 3650212 3680374 . / . TIER IV, Inc., [28]“Autoware扩散规划器”，https://github. com/autowarefoundation/autoware universe/tree/main/planning/ autoware diffusion planner，2025年。[29] C. M. Jiang、Y. Bai、A. Cornman、C. Davis、X. Huang、H. Jeon、S. Kulshrestha、J. Lambert、S. Li、X. Zhou、C. Fuertes、C. Yuan、M. Tan、Y. Zhou 和 D. Anguelov，“SceneDiffuser：高效且可控的驾驶仿真初始化与推演”，载于《神经信息处理系统进展》第37卷，A. Globerson、L. Mackey、D. Belgrave、A. Fan、U. Paquet、J. Tomczak 和 C. Zhang 编，第37卷。美国加利福尼亚州圣迭戈：神经信息处理系统基金会（NeurIPS），2024年，第55 729–55 760页。 [在线]。获取地址：http://dx.doi.org/10.52202/079017-1771