Diffusion-ES: Gradient-free Planning with Diffusion for Autonomous and Instruction-guided Driving

Brian Yang Huangyuan Su Nikolaos Gkanatsios Tsung-Wei Ke
Ayush Jain Jeff Schneider Katerina Fragkiadaki
Carnegie Mellon University

1. Model training details

We train our diffusion models on 1% of the released nuPlan training data, which is subsampled from the original 20Hz to 0.5Hz. Our diffusion models are DDIMs [3] trained with $T = 100$ diffusion steps. We use the scaled linear beta schedule and predict ϵ. Our models are implemented in PyTorch and we use the HuggingFace Diffusers library to implement our diffusion models.

Our base diffusion architecture is as follows. Each trajectory waypoint is linearly projected to a latent feature with hidden size 256. The noise level is encoded with sinusoidal positional embeddings followed by a 2-layer MLP. Noise features are fused with the trajectory tokens by concatenating the noise feature to all trajectory features along the feature dimension and projecting back to the hidden size of 256. We also apply rotary positional embeddings [4] to the trajectory tokens as temporal embeddings. We then pass all the trajectory tokens through 8 transformer encoder layers, and each trajectory token is decoded to a corresponding waypoint. The final trajectory consists of the stacked waypoint predictions. Our conditional diffusion policy baseline uses a similar architecture, except we featurize the scene using the backbone from the nuPlan re-implementation of Urban Driver [2] and pass those tokens into the self-attention layers.

We train our models with batch size 256 and use the AdamW optimizer with learning rate 1e-4, weight decay 5e-4, and $(\beta_1, \beta_2) = (0.9, 0.999)$.

Our trajectories consist of 16 2D pose waypoints each with 3 features (x, y, θ). We preprocess these trajectory features by applying Verlet wrapping as described in MotionLM [**?**], which we found to improve performance by encouraging smooth trajectories.

2. Language instruction following tasks

Here, we describe in detail each of the controllability tasks. For each, we list the task goal as well as the specific language instruction used.

1. **Lane change**: the ego-vehicle must execute a lane change. The language instruction is *"Change lanes to the left"*. The episode is considered a success if the ego-vehicle reaches the left lane.
2. **Unprotected left turn**: the ego-vehicle must perform an unprotected left turn. The language instruction is *"If car 18 is within 20 meters yield to it. Otherwise it will slow for you"*, where car 18 is the incoming car. The episode is considered a success if the ego-vehicle either completes the turn before the incoming car, or the incoming car passes the ego freely indicating a successful yield. Due to the randomized agent behaviors, it is not always possible to safely execute the turn in this task.
3. **Unprotected right turn**: the ego-vehicle must perform an unprotected right turn. The language instruction is *"Change to lane 33."*, where lane 33 is the target lane. The episode is considered a success if the ego-vehicle completes the right turn.
4. **Overtaking**: the ego-vehicle must overtake the target car. The language instruction is "Car 21 will slow for you. Change to the right lane. Once ahead of car 4 change to the left lane.", where car 21 is the incoming car in the right lane and car 4 is the target car. The episode is considered a success if the ego-vehicle is ahead of the target car while in the same lane.
5. **Extended overtaking**: the ego-vehicle must overtake the target car across several lanes of dense traffic. The language instruction is *"Change two lanes to the left. Then if you are ever ahead of car 3 change lanes to the right"*. The episode is considered a success if the ego-vehicle is ahead of the target car while in the same lane.
6. **Yielding**: the ego-vehicle must allow a car approaching quickly from behind to pass by changing lanes. The language instruction is *"Slow down and change lanes to the left. Then once car 8 is ahead of you change lanes to the right."*, where car 8 is the incoming car. The episode is considered a success if the ego-vehicle is behind the target car.
7. **Cut in**: the ego-vehicle must cut in to a column of cars. The language instruction is *"Slow down. Car 2 will slow for you. Change two lanes to the left"*, where car 2 is the car we are cutting in front of. The episode is considered a

success if the ego-vehicle is in front of the target car and in the same lane.

8. **Lane weaving**: the ego-vehicle must reach a specific gap between two cars across several lanes of dense traffic. The language instruction is *"Once ahead of car 12 change lanes to the left. Then slow down a lot and change lanes to the left. Once car 9 is ahead of you by a few meters change lanes to the left"*. The episode is considered a success if the ego-vehicle successfully reaches the target gap.

3. Language instruction following prompts

We collect 24 instruction-program pairs in total, and they are shown in full below. To organize and manage our prompts, we use DSPy [1].

Listing 1. Full prompt with all instruction-program pairs

```
Use the provided language instruction to write code for guiding a lower-level driving policy.

---

Follow the following format.

Instruction: ${instruction}
Code: ${code}

---

Instruction: Yield to car 4. If car 2 is ahead of car 4 stop yielding.
Code:
done = self.yield_to_vehicle(self.get_vehicle(4))
def vehicle_ahead_of_other_vehicle():
    vehicle = self.get_vehicle(2)
    other_vehicle = self.get_vehicle(4)
    return vehicle.is_ahead_of(other_vehicle)
while not done() and not vehicle_ahead_of_other_vehicle():
    yield
self.stop_yielding()

---

Instruction: If car 1 is not stopped yield to it.
Code:
vehicle = self.get_vehicle(1)
if not vehicle.is_stopped():
    done = self.yield_to_vehicle(vehicle)
    while not done():
        yield
    self.stop_yielding()

---

Instruction: Change to the left lane.
Code:
done = self.follow_lane(self.left_lane)
while not done():
    yield

---

Instruction: If the speed of car 2 is lower than the speed limit change to the right lane.
Code:
vehicle = self.get_vehicle(2)
if vehicle.speed < self.current_lane.speed_limit:
    done = self.follow_lane(self.right_lane)
    while not done():
        yield

---

Instruction: Car 20 will slow down.
Code:
vehicle = self.get_vehicle(20)
self.adjust_constant_velocity_prediction(vehicle, 0.5)

---

Instruction: If car 17 is within 20 meters yield to it.
Code:
vehicle = self.get_vehicle(17)
if vehicle.distance_to(self.ego_vehicle) < 20.0:
    done = self.yield_to_vehicle(vehicle)
    while not done():
        yield
    self.stop_yielding()

---

Instruction: Resume normal driving speed.
Code: self.unset_ego_speed_limit()

---

Instruction: While you are ahead of car 3 stay in the current lane. Otherwise change to their lane.
Code:
```

```
def ahead_of_vehicle():
    vehicle = self.get_vehicle(3)
    return self.ego_vehicle.is_ahead_of(vehicle)
while ahead_of_vehicle():
    self.follow_lane(self.current_lane)
    yield
vehicle = self.get_vehicle(3)
their_lane = vehicle.get_closest_lane(self.lane_graph)
done = self.follow_lane(their_lane)
while not done():
    yield
```

Instruction: Follow lane 12.
Code:
```
lane = self.get_lane(12)
done = self.follow_lane(lane)
while not done():
    yield
```

Instruction: Once the speed of car 2 is lower than the speed limit of your current lane change to the right lane.
Code:
```
speed_limit = self.current_lane.speed_limit
def speed_exceeds_limit():
    vehicle = self.get_vehicle(2)
    return vehicle.speed < speed_limit
while speed_exceeds_limit():
    yield
done = self.follow_lane(self.right_lane)
while not done():
    yield
```

Instruction: Change to lane 1.
Code:
```
lane = self.get_lane(1)
done = self.follow_lane(lane)
while not done():
    yield
```

Instruction: Yield to car 3.
Code:
```
vehicle = self.get_vehicle(3)
done = self.yield_to_vehicle(vehicle)
while not done():
    yield
self.stop_yielding()
```

Instruction: If car 2 is ever ahead of you by 10 meters it will slow for you.
Code:
```
def vehicle_ahead_of_us():
    vehicle = self.get_vehicle(2)
    return vehicle.is_ahead_of(self.ego_vehicle, 10.0)
while not vehicle_ahead_of_us():
    yield
self.set_velocity_ratio(self.get_vehicle(2), 0.5)
```

Instruction: Change lanes to the right. Then change lanes to the left.
Code:
```
done = self.follow_lane(self.right_lane)
while not done():
    yield
done = self.follow_lane(self.left_lane)
while not done():
    yield
```

Instruction: If the speed of car 8 is greater than 5.0 yield to it.

```
Code:
vehicle = self.get_vehicle(8)
if vehicle.speed > 5.0:
    done = self.yield_to_vehicle(vehicle)
    while not done():
        yield
    self.stop_yielding()

---

Instruction: Change lanes to the right. Once car 1 is ahead of you yield to it.
Code:
done = self.follow_lane(self.right_lane)
while not done():
    yield
def vehicle_ahead_of_us():
    vehicle = self.get_vehicle(1)
    return vehicle.is_ahead_of(self.ego_vehicle)
while not vehicle_ahead_of_us():
    yield
done = self.yield_to_vehicle(self.get_vehicle(1))
while not done():
    yield
self.stop_yielding()

---

Instruction: Slow down and change lanes to the right.
Code:
current_speed = self.ego_vehicle.speed
self.set_ego_speed_limit(current_speed * 0.5)
done = self.follow_lane(self.right_lane)
while not done():
    yield

---

Instruction: Slow down a lot and change lanes to the right. Then once car 2 is ahead of you, resume normal driving
     speed.
Code:
current_speed = self.ego_vehicle.speed
self.set_ego_speed_limit(current_speed * 0.2)
done = self.follow_lane(self.right_lane)
while not done():
    yield
def vehicle_ahead_of_us():
    vehicle = self.get_vehicle(2)
    return vehicle.is_ahead_of(self.ego_vehicle)
while not vehicle_ahead_of_us():
    yield
current_speed = self.ego_vehicle.speed
self.unset_ego_speed_limit()

---

Instruction: Stay in the current lane. If your speed exceeds 2.0 stop following the current lane.
Code:
done = self.follow_lane(self.current_lane)
def speed_under_threshold():
    speed = self.ego_vehicle.speed
    return speed < 2.0
while not done() and speed_under_threshold():
    yield
self.stop_following()

---

Instruction: While car 3 is ahead of you stay in the current lane. Otherwise change to their lane.
Code:
def vehicle_ahead_of_us():
    vehicle = self.get_vehicle(3)
    return vehicle.is_ahead_of(self.ego_vehicle)
while vehicle_ahead_of_us():
    self.follow_lane(self.current_lane)
    yield
vehicle = self.get_vehicle(3)
their_lane = vehicle.get_closest_lane(self.lane_graph)
done = self.follow_lane(their_lane)
while not done():
```

```
    yield

---

Instruction: If there is a car in the left lane going faster than 5.0 m/s stay in the current lane.
Code:
vehicles_in_left_lane = self.left_lane.get_vehicles()
if any([vehicle.speed > 5.0 for vehicle in vehicles_in_left_lane]):
    done = self.follow_lane(self.current_lane)
    while not done():
        yield

---

Instruction: If car 2 is ahead of you by 10 meters it will slow for you.
Code:
vehicle = self.get_vehicle(2)
if vehicle.is_ahead_of(self.ego_vehicle, 10.0):
    self.adjust_constant_velocity_prediction(vehicle, 0.5)

---

Instruction: Stay in the current lane.
Code:
done = self.follow_lane(self.current_lane)
while not done():
    yield

---

Instruction: If car 20 is ever stopped yield to it.
Code:
def vehicle_is_ever_stopped():
    vehicle = self.get_vehicle(20)
    return vehicle.is_stopped()
while not vehicle_is_ever_stopped():
    yield
vehicle = self.get_vehicle(20)
done = self.yield_to_vehicle(vehicle)
while not done():
    yield
self.stop_yielding()

---

Instruction: [insert command here]
Code:
```

References

[1] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023. 2

[2] Oliver Scheel, Luca Bergamini, Maciej Wolczyk, Błażej Osiński, and Peter Ondruska. Urban driver: Learning to drive from real-world demonstrations using policy gradients. In *Conference on Robot Learning*, pages 718–728. PMLR, 2022. 1

[3] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020. 1

[4] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021. 1