A Details of Q-former based Vision Language Embedding Module

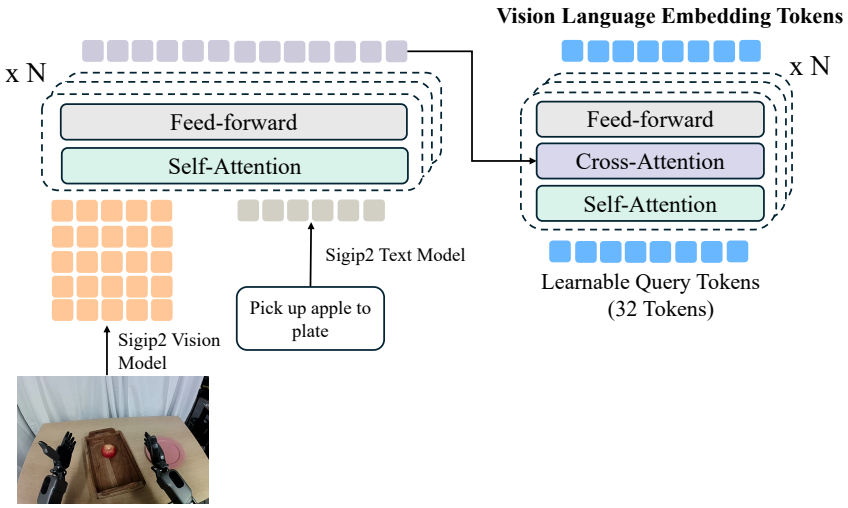


Figure 9: Our Q-former based Vision Language Embedding Module

We present the architectural details of our compact Q-former-based vision-language embedding module. Specifically, we adopt `siglip2-large-patch16-256` as the backbone for both vision and language encoders. The SigLIP2 vision encoder processes 256×256 resolution robot images into 256 patch tokens, while the language encoder encodes padded robot instructions into 32 language tokens. hese 256 vision tokens and 32 language tokens are concatenated and passed through four layers of self-attention transformers to yield 288 fused vision-language tokens. To obtain a compact representation, we apply a Q-former architecture [15], where 32 learnable query tokens—randomly initialized—interact with the 288 fused tokens through interleaved self-attention and cross-attention layers, producing 32 compressed vision-language tokens.

B Pretraining Data Mixture

Details of pretraining data mixture is presented in Table 3.

Table 3: Action-Aware Vision Language Embedding Pre-training Dataset Statistics

Dataset	Length (Frames)	Duration (hr)	FPS	Camera View	Category
GR-1 In-house Dataset	6.4M	88.4	20	Egocentric	Real robot
DROID (OXE) [28]	23.1M	428.3	15	Left, Right, Wrist	Real robot
RT-1 (OXE) [29]	3.7M	338.4	3	Egocentric	Real robot
Language Table (OXE) [30]	7.0M	195.7	10	Front-facing	Real robot
Bridge-v2 (OXE) [31]	2.0M	111.1	5	Shoulder, left, right, wrist	Real robot
MUTEX (OXE) [32]	362K	5.0	20	Wrist	Real robot
Plex (OXE) [33]	77K	1.1	20	Wrist	Real robot
RoboSet (OXE) [34]	1.4M	78.9	5	Left, Right, Wrist	Real robot
GR-1 Simulation	125.5M	1,742.6	20	Egocentric	Simulation
Total	169.5M	2,989.5	–	–	–

C Training Details

For the pretraining of action-aware vision language embedding module, we use 256 NVIDIA H100 GPUs with batch size 8192 for 150,000 gradient steps. We use AdamW [35] optimizer with $\beta_1 = 0.95, \beta_2 = 0.999$, and $\epsilon =$ 1e-8. A weight decay of 1e-5 is applied, and the learning rate

follows a cosine scheduling strategy with a warmup ratio of 0.05. Following [2, 3], we sample the flowmatching denoising timestep from $p(\tau) = \text{Beta}(\frac{s-\tau}{s}; 1.5, 1)$, $s = 0.999$.

For the multitask experiments of FLARE conducted in Section 4.1 and 4.2, we use 32 NVIDIA H100 GPUS with batch size 1024 for 80,000 gradient steps, while keeping the rest of the hyperparameter setups exactly the same.

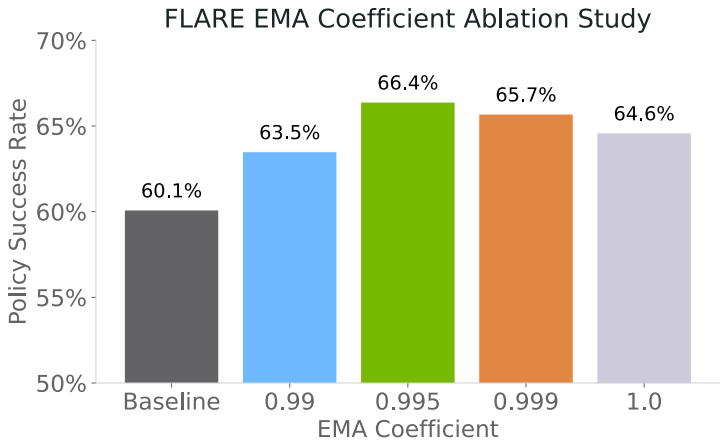


Figure 10: Effect of EMA Coefficient ρ: We report the policy success rate using 24×300 training trajectories across 24 RoboCasa tasks. Baseline is trained without FLARE future alignment loss, i.e. a policy only objective.

D Exponential Moving Average (EMA) of Pretrained Action-aware Embedding Model

As discussed in Section 3.2, to address the distribution shift between pretraining and downstream tasks for our action-aware vision-language target embedding model, we incorporate an exponential moving average (EMA) update. Specifically, at each gradient step, the target embedding model parameters are updated as follows:

$$\theta_{\text{target\_vl\_embedding}} \leftarrow \rho\theta_{\text{target\_embedding}} + (1-\rho)\theta_{\text{policy\_vl\_embedding}}$$

While the policy's vision-language encoder is initialized from the target vision-language encoder, i.e. pretrained action-aware vision-language embedding, it gradually adapts to the downstream task during training via the action flow-matching objective. The EMA update enables the prediction target to adapt slowly in tandem with the evolving policy encoder, providing stability across training.

We evaluate several choices of the EMA coefficient $\rho \in \{0.99, 0.995, 0.999, 1.0\}$, each using 24×300 trajectories to train the FLARE policy. The final average success rates are reported in Figure 10. We find that while all EMA variants outperform the baseline method without FLARE future latent alignment objective, $\rho = 0.995$ yields the best performance and is used in all experiments. Notably, even with $\rho = 1.0$ (i.e., no EMA), FLARE still surpasses the baseline, whereas $\rho = 0.99$ performs the worst, likely due to the instability caused by frequent target updates.

E Pseudocode of FLARE

Here we present a Python-style pseudocode of FLARE loss calculation as well as the entire training loop.

Algorithm 1 Python-style pseudocode for FLARE training

```
# target_vl_embedding: pretrained action-aware vision language embedding
# vl_embedding:  vision language embedding of the current policy
# dit: diffusion transformer of the current policy
# action_embedding: 2-layer MLP to embed noisy actions
# state_embedding:  2-layer MLP to embed prioprioceptive state
# action_decode:    2-layer MLP to decode robot's actions
# embedding_decode: 2-layer MLP to decode predicted embeddings
# N: Number of gradient steps
# M: Number of tokens in VL
# lambda: coefficient of FLARE loss (default is 0.2)

### Initialization
future_tokens = nn.Embedding(M, hiddem_dim)
vl_embedding.load_state_dict(vl_embedding.state_dict())
target_vl_embedding.requires_grad = False

for n in range(N):
    obs, proprio, actions, future_obs = dataset.next()

    ### Prepare noisy action inputs
    noise = gaussian.sample()
    timestep = beta.sample() # sample flowmatching timestep
    noisy_action = timestep * actions + (1-timestep) * noise
    velocity = actions - noise

    ### Get state, action, and observation embedding tokens
    action_tokens = action_embed(noisy_action, timestep)
    state_token   = state_embed(state)
    vl_tokens     = vl_embedding(obs)

    ### Pass through DiT layers
    sa_tokens  = torch.concat([state_token, action_tokens, future_tokens], dim=1)
    policy_outputs = dit(sa_tokens, vl_tokens)

    ### Calculate action flowmatching loss
    action_outputs = action_decoder(policy_outputs[:, 1:1 + action_tokens.shape[1]])
    action_loss = MSE(action_outputs, velocity)

    ### Calculate FLARE loss
    with torch.no_grad():
        embedding_to_align = target_vl_embedding(future_obs)
    predict_embedding = decode_embedding(policy_outputs[:, -M:])
    flare_loss = 1-COSINE_SIMILARITY(predict_embedding, embedding_to_align)

    ### Optimize the combined loss
    loss = action_loss + lambda * flare_loss
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

F Real GR1 Humanoid Rollouts

F.1 4 Pick-and-place Tasks

Below, we present policy rollouts from the FLARE trained policy on 4 real-world GR1 humanoid pick-and-place tasks, together with the task's language instructions. Qualitatively, we observe that when manipulating objects such as bottled water or coke can, the FLARE policy learns to maneuver the hand around the object, climbing overt the water bottle, rather than striking and knocking it over. For live 1x playback videos of the robot executions, please refer to the accompanying 3-minute video presentation.

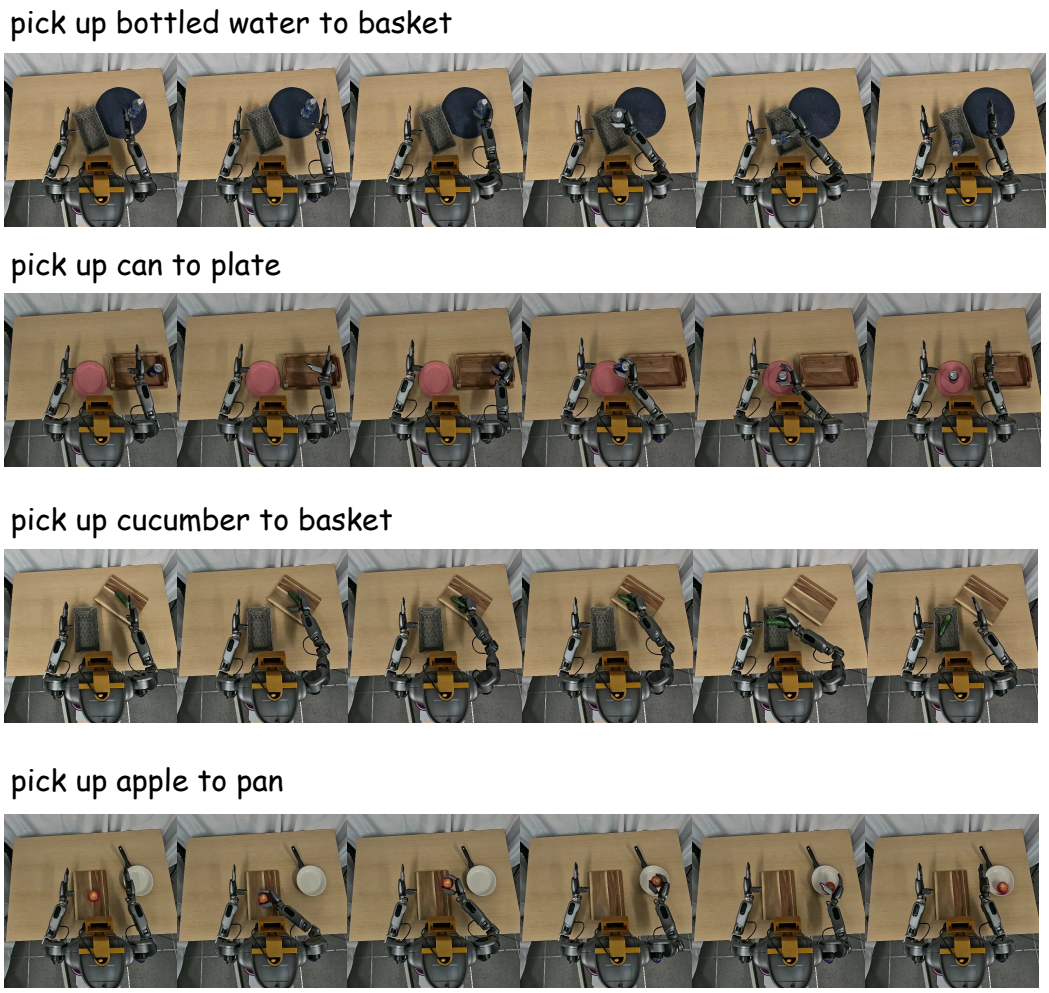
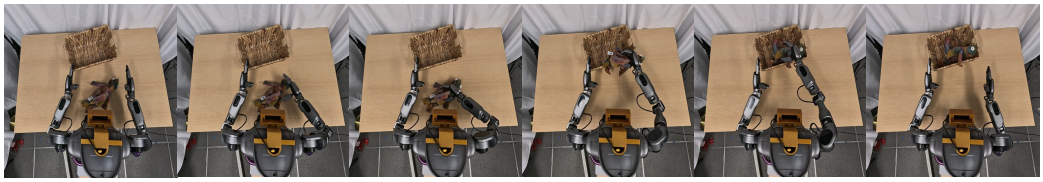


Figure 11: FLARE policy rollout on real GR1 humanoid robot with 4 pick-and-place tasks

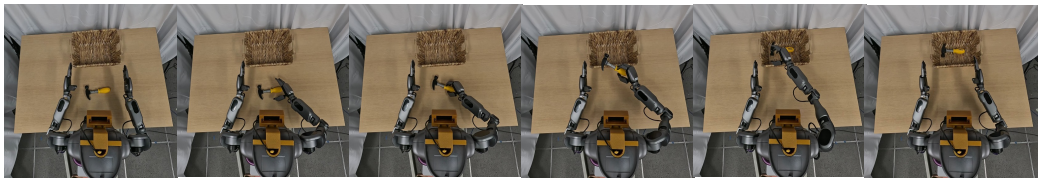
F.2 Manipulating Novel Objects

Below, we present policy rollouts from the FLARE trained policy manipulating 5 novel objects. For live 1x playback videos of the robot executions, please refer to the accompanying 3-minute video presentation.

pick up stuffed toy to basket



pick up hammer to plate



pick up blue tape to basket

pick up blackboard eraser to pan

pick up umbrella to pan

Figure 12: FLARE policy rollout manipulating 5 novel objects