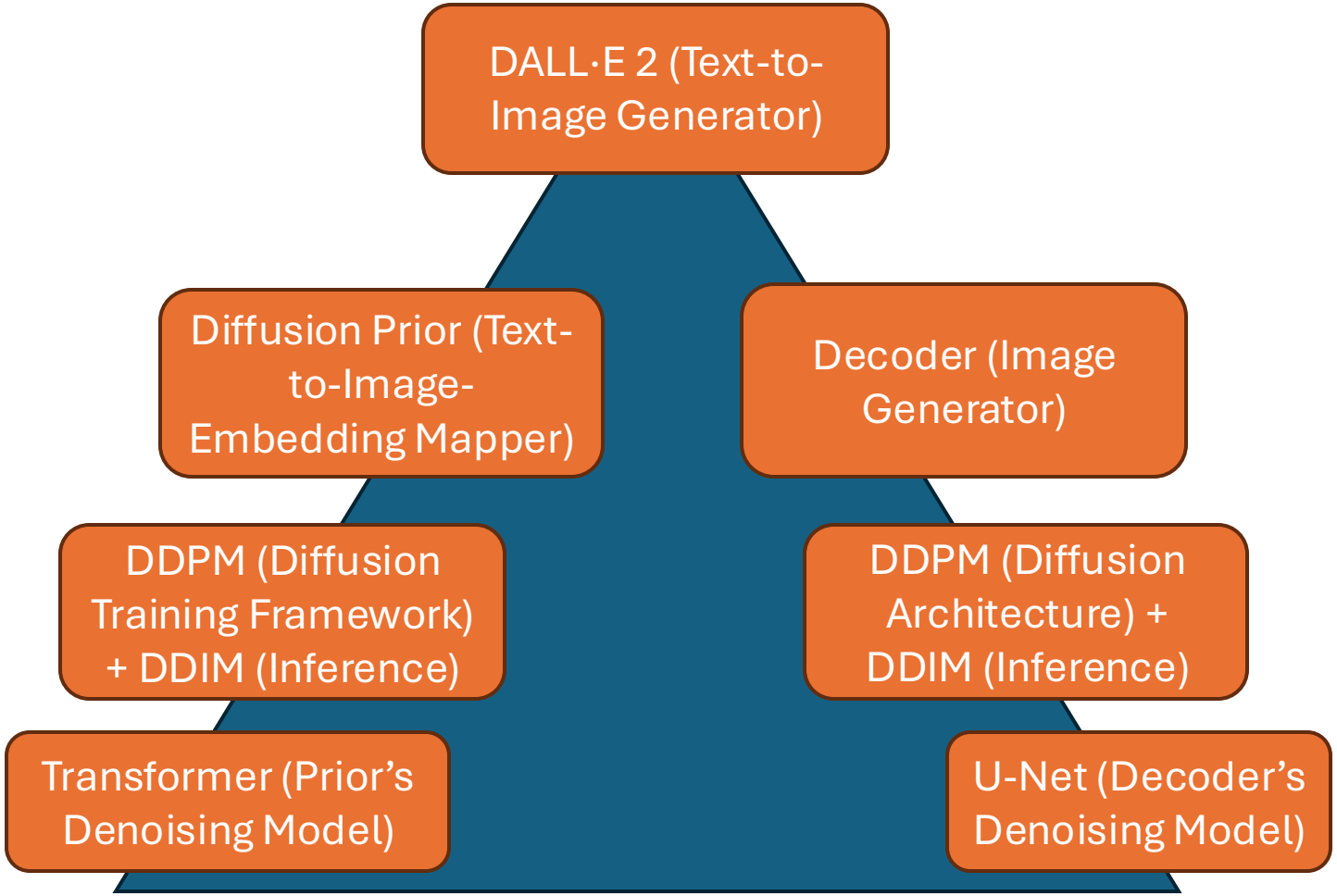
DALL·E 2-AWS Project: DALL·E 2 Planning

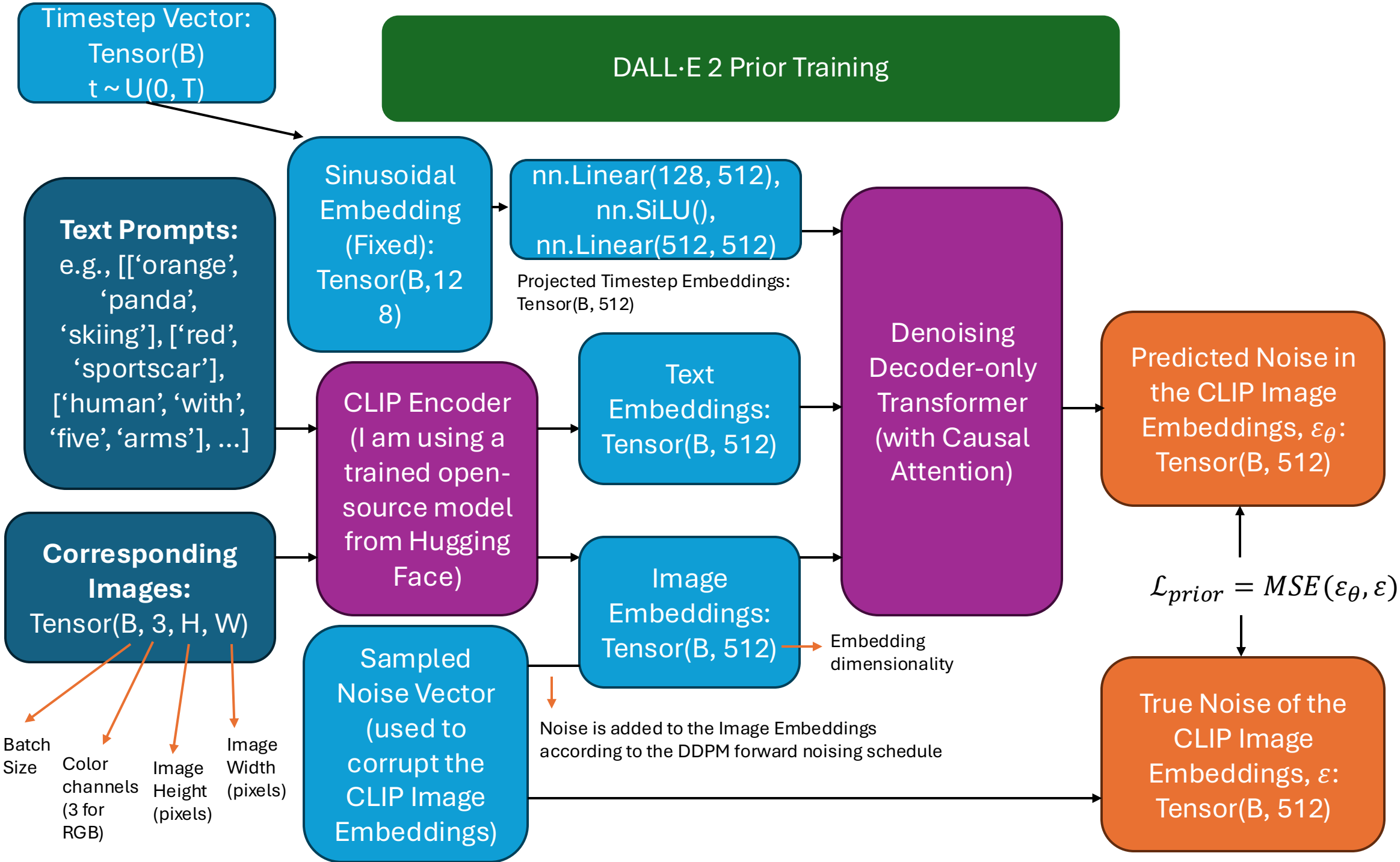
Spencer Karofsky

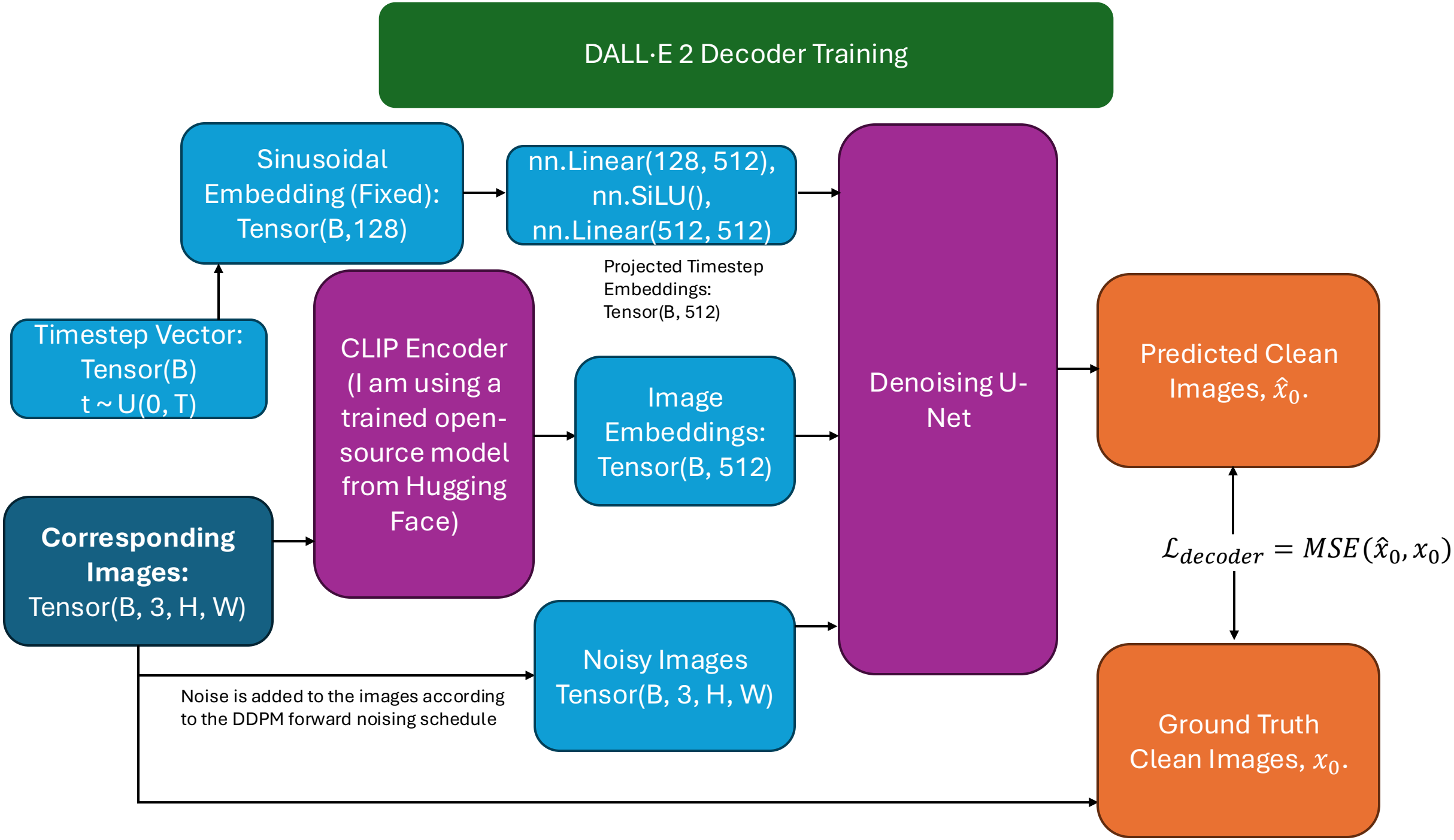
DALL·E 2 High Level Architecture Pyramid

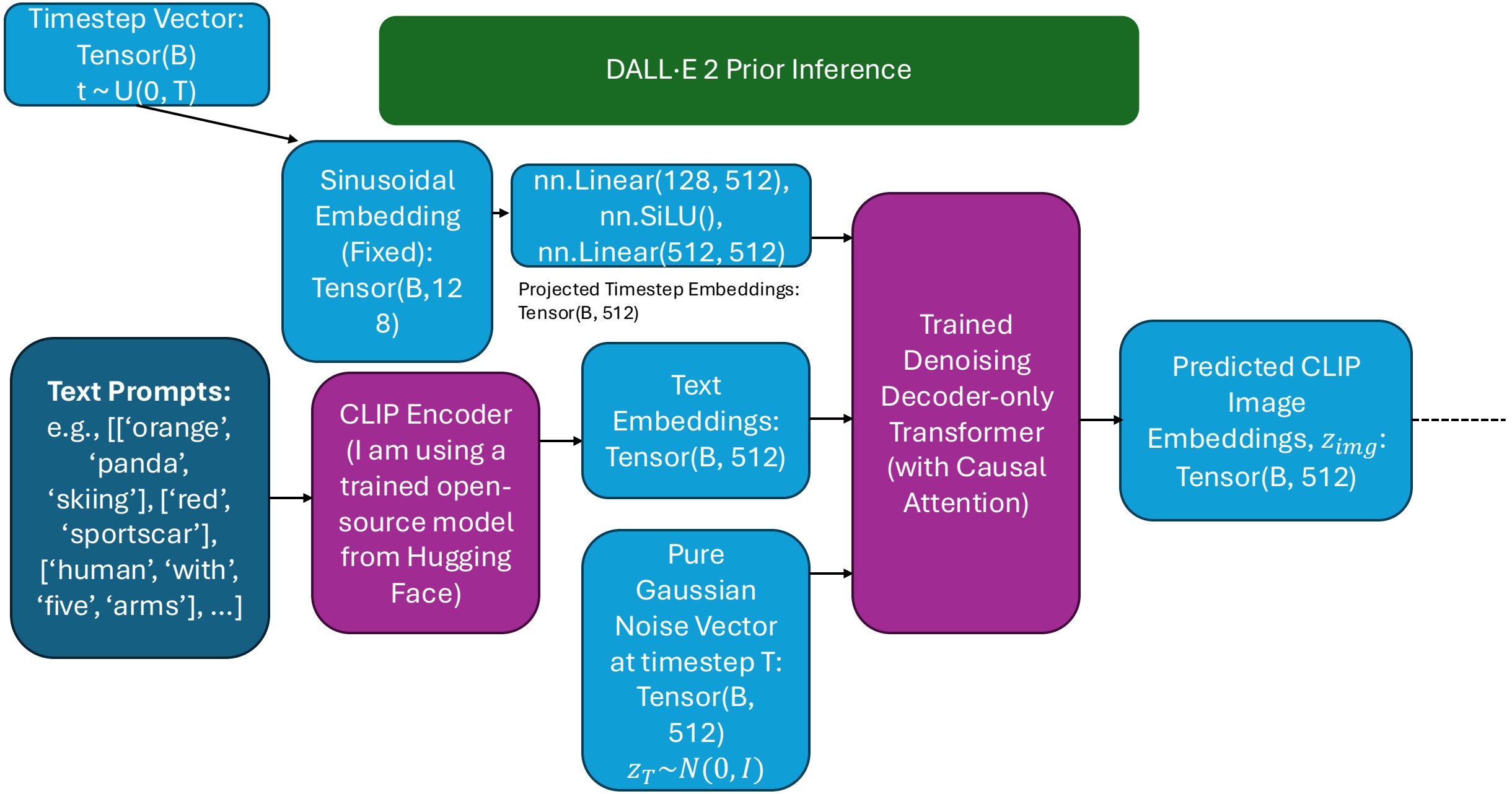


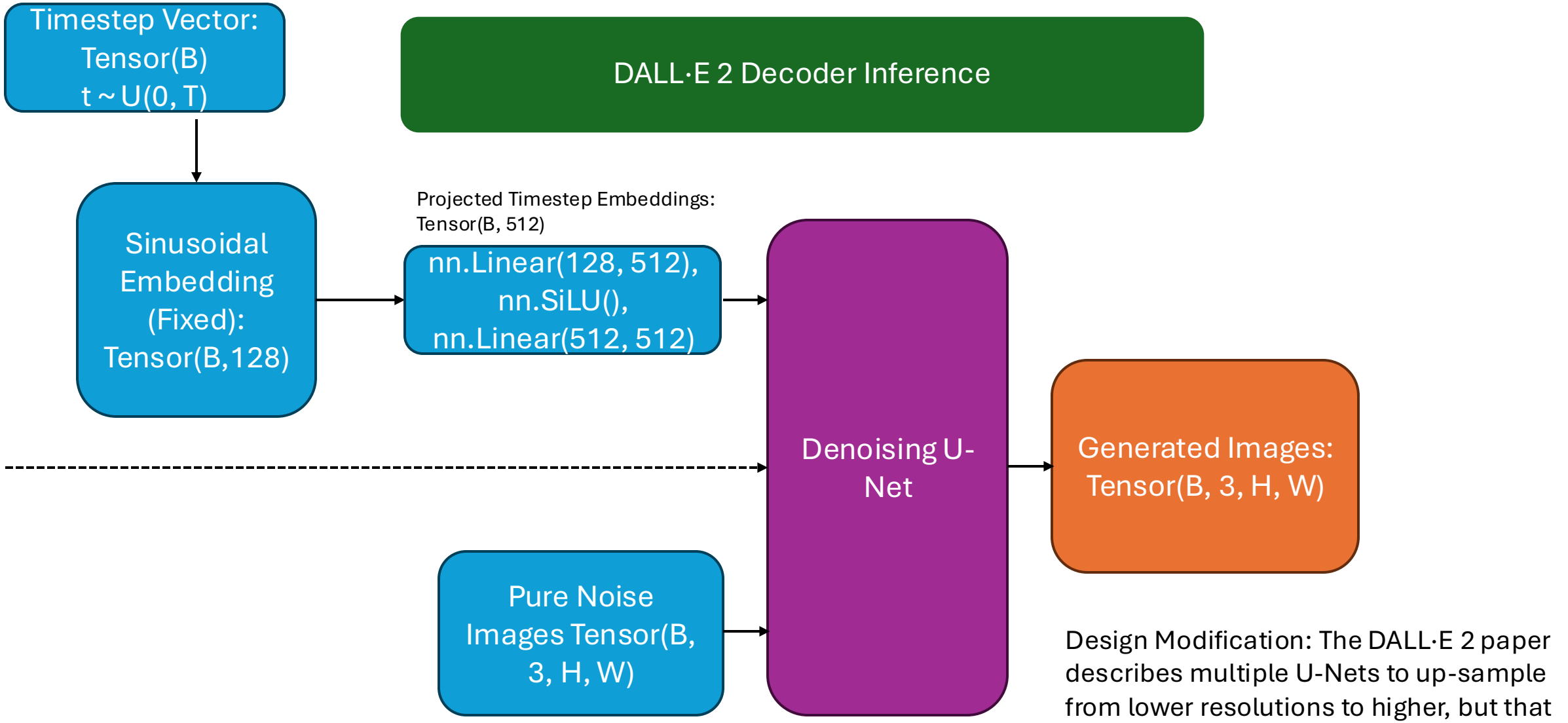
Conceptual Overview

Training and Inference Pipelines


DALL·E 2 Prior Training
Timestep Vector: Tensor(B) t ~ U(0, T)
Sinusoidal Embedding (Fixed): Tensor(B,128)
nn.Linear(128, 512), nn.SiLU(), nn.Linear(512, 512)
Projected Timestep Embeddings: Tensor(B, 512)
Text Prompts: e.g., [['orange', 'panda', 'skiing'], ['red', 'sportscar'], ['human', 'with', 'five', 'arms'], ...]
CLIP Encoder (I am using a trained open-source model from Hugging Face)
Text Embeddings: Tensor(B, 512)
Corresponding Images: Tensor(B, 3, H, W)
Image Embeddings: Tensor(B, 512)
Embedding dimensionality
Denoising Decoder-only Transformer (with Causal Attention)
Predicted Noise in the CLIP Image Embeddings, ε_θ: Tensor(B, 512)
$\mathcal{L}_{prior} = MSE(\varepsilon_\theta, \varepsilon)$
Batch Size
Color channels (3 for RGB)
Image Height (pixels)
Image Width (pixels)
Sampled Noise Vector (used to corrupt the CLIP Image Embeddings)
Noise is added to the Image Embeddings according to the DDPM forward noising schedule
True Noise of the CLIP Image Embeddings, ε: Tensor(B, 512)


DALL·E 2 Decoder Training
Sinusoidal Embedding (Fixed): Tensor(B,128)
nn.Linear(128, 512), nn.SiLU(), nn.Linear(512, 512)
Projected Timestep Embeddings: Tensor(B, 512)
Timestep Vector: Tensor(B) t ~ U(0, T)
CLIP Encoder (I am using a trained open-source model from Hugging Face)
Image Embeddings: Tensor(B, 512)
Denoising U-Net
Predicted Clean Images, $\hat{x}_0$.
$\mathcal{L}_{decoder} = MSE(\hat{x}_0, x_0)$
Corresponding Images: Tensor(B, 3, H, W)
Noise is added to the images according to the DDPM forward noising schedule
Noisy Images Tensor(B, 3, H, W)
Ground Truth Clean Images, x_0.


Timestep Vector: Tensor(B) t ~ U(0, T)
DALL·E 2 Prior Inference
Sinusoidal Embedding (Fixed): Tensor(B,128)
nn.Linear(128, 512), nn.SiLU(), nn.Linear(512, 512)
Projected Timestep Embeddings: Tensor(B, 512)
Text Prompts: e.g., [['orange', 'panda', 'skiing'], ['red', 'sportscar'], ['human', 'with', 'five', 'arms'], ...]
CLIP Encoder (I am using a trained open-source model from Hugging Face)
Text Embeddings: Tensor(B, 512)
Pure Gaussian Noise Vector at timestep T: Tensor(B, 512) $z_T \sim N(0, I)$
Trained Denoising Decoder-only Transformer (with Causal Attention)
Predicted CLIP Image Embeddings, z_{img}: Tensor(B, 512)



Design Modification: The DALL·E 2 paper describes multiple U-Nets to up-sample from lower resolutions to higher, but that choice is designed for 1024x1024 images. I only use one U-Net as I am generating a maximum image resolution of either 128x128 or 256x256.

DALL·E 2 High Level Architecture Description

- DALL·E 2: proposed text-to-image architecture by OpenAI that contains two trainable components: a prior and decoder.
  - Prior: Learns to generate image embeddings conditioned on the text embedding and timestep.
    - Training: A Transformer that takes in a noisy CLIP image embedding (real image embedding with noise added as defined according to the DDPM forward noising process), a clean CLIP text embedding, and timestep index (randomly sampled) and learns to either predict the noise in the CLIP image embedding or the denoised image embedding (either output type works; explained more in future slides).
    - Inference: The trained Transformer that takes in a pure Gaussian noise vector, the same (frozen) CLIP text embedding conditioned on the input caption, and the time step we are sampling from (decrementing from T to 0) that learns to transform random noise into a semantically-meaningful image embedding via a diffusion process.
  - Decoder: Learns to reconstruct an image from random noise, conditioned on a CLIP image embedding and timestep.
    - Training: A U-Net that takes in real (encoded) images that have been noised according to the DDPM forward noising process, CLIP image embeddings, and timesteps and learns (through a diffusion process) to predict a clean image or the noise (either output type works; explained more in future slides).
    - Inference: The trained U-Net that takes in pure Gaussian noise, the predicted image embedding (from the prior), and the timestep and outputs a final generated image.
- CLIP encoder model–a shared, fixed/non-trainable, and high-dimensional feature space between the prior and decoder.
- Important Note: During training, we train the prior and decoder separately (with different training objectives); we only use them together during inference, when the prior's output conditions the decoder.

Implementation: Phase I

Building an MVP

class TimestepEmbedder(nn.Module):

- Input: $dim = 512$
- Methods:
  - forward($t \in \mathbb{R}^{(B,1)}$) -> $t_{emb} \in \mathbb{R}^{(B,dim)}$
- Used by Prior, Decoder

class CLIPEncoder:

- Input: None; I am using a trained CLIP Encoder from OpenCLIP with an embedding dimensionality of 512.
- Methods:
  - encode_text(list of N text prompts (strings)) -> $z_{txt} \in \mathbb{R}^{(N,512)}$
  - encode_images(list of N images (List[PIL.Image])) -> $z_{img} \in \mathbb{R}^{(N,512)}$
  - @property; dim() -> 512 (for ease of use for other classes to query dimensionality)

class NoiseScheduler:

- Input: $T = 1000$
- Methods:
  - get_beta_t($t \in \mathbb{Z}^{(B)}, \eta$) -> $\beta_t \in \mathbb{R}^{(B)}$
  - get_alpha_t($t \in \mathbb{Z}^{(B)}, \eta$) -> $\alpha_t \in \mathbb{R}^{(B)}$
  - get_alpha_bar_t($t \in \mathbb{Z}^{(B)}, \eta$) -> $\bar{\alpha}_t \in \mathbb{R}^{(B)}$

class DDIMSampler:

- Input: noise scheduler (NoiseScheduler)
- Methods:
  - sample($model, z_{cond} \in \mathbb{R}^{(B,512)}, shape_{output}, steps \in \mathbb{Z}$) -> $x_t \in \mathbb{R}^{shape}$

class Prior(nn.Module):

- Inputs: $T \in \mathbb{Z}$
- Methods:
  - forward(z_{txt}, x_t, t_{emb}) -> $\varepsilon_\theta \in \mathbb{R}^{(B,\,512)}$ (predicted CLIP embedding noise)
    - One denoising step
  - sample(z_{txt}, t) -> $\hat{z}_{img} \in \mathbb{R}^{(B,\,512)}$ (predicted CLIP image embedding)
    - Full denoising process

class Decoder(nn.Module):

- Inputs: $T \in \mathbb{Z}$
- Methods:
  - forward($x_{t\,,img}, z_{img}, t_{emb}$) -> $\varepsilon_\theta \in \mathbb{R}^{(B,3,H,W)}$ (predicted noise in pixel space)
  - sample($z_{img}, steps, eta \in [0,1)$) -> $x_0 \in \mathbb{R}^{(B,3,H,W)}$ (predicted clean image in pixel space)

class PriorTransformer(nn.Module):

- Inputs: $dim = 512, depth = 6, heads = 8$
- Methods:
  - forward(z_{txt}, z_t, t_{emb}) ->$\varepsilon_\theta \in \mathbb{R}^{(B,\,512)}$ (predicted CLIP embedding noise)

class DecoderUNet(nn.Module):

- Inputs: $z_{t,img}, z_{img}, t_{emb}$
- Methods:
  - forward($x_{t\,,img}, z_{img}, t_{emb}$) ->$\varepsilon_\theta \in \mathbb{R}^{(B,3,H,W)}$ (predicted noise in pixel space)

class DALLe2:

- Inputs: prior_path (string), decoder_path (string), CLIP encoder (CLIPEncoder), prior_sampler (DDIMSampler for latent), decoder_sampler (DDIMSampler for pixel space)
- Methods:
  - generate(prompts (list of N string prompts), # of steps (int), $\eta \in [0,1)$) -> $\hat{x}_0 \in \mathbb{R}^{(N,3,H,W)}$
  - _encode_text(prompts (list of N string prompts)) -> $z_{txt} \in \mathbb{R}^{(N,512)}$
  - _encode_images(images (list of N PIL Images)) -> $z_{img} \in \mathbb{R}^{(N,512)}$

class DALLe2Trainer:

- Inputs: model (Prior or Decoder), model_type ("prior" or "decoder"), optimizer (torch.optim.Optimizer()), noise_scheduler (NoiseScheduler), sampler (DDIMSampler), dataloader (torch.utils.data.DataLoader), is_aws (bool = False), save_dir (str), log_interval (int)
- Methods:
  - train(num_epochs (int)) -> None
  - train_one_epoch(epoch_i (int)) -> None
  - train_step(batch (dict)) -> loss (float)
    - batch (prior): $\{z_t \in \mathbb{R}^{(B,512)}, z_{txt} \in \mathbb{R}^{(B,512)}, t \in \mathbb{Z}^{(B)}, \varepsilon \in \mathbb{R}^{(B,512)}\}$
    - batch (decoder): $\{x_t \in \mathbb{R}^{(B,3,H,W)}, z_{img} \in \mathbb{R}^{(B,512)}, t \in \mathbb{Z}^{(B)}, x_0 \in \mathbb{R}^{(B,3,H,W)}\}$

Directory Structure (for Maximum Organization)

- dalle_2_new/ # Once complete, change to dalle_2/
  - models/
    - dalle2.py
    - prior.py
    - prior_transformer.py
    - decoder.py
    - decoder_unet.py
    - timestep_embedding.py
    - clip_encoding.py
  - sampling/
    - ddim_sampler.py
    - noise_scheduler.py
  - training/
    - train_prior.py
    - train_decoder.py
    - trained_models/
      - *.pth
  - data/
    - abs_dataset.py
    - [dataset name 1].py
    - [dataset name 2].py
    - …
    - [dataset name N].py
    - dataset.py
  - inference/
    - generate.py
    - generations/
      - *.jpg