Appendix

Pseudo-code for calculating $\overline{\text{FID}}_\infty$. Evaluating $\overline{\text{IS}}_\infty$ is exactly the same except swapping the FID calculation function for IS calculation function.

Algorithm 1 Evaluating $\overline{\text{FID}}_\infty$ for a Generator

G: Generator
I: Inception Network
n: Number of images we want to generate
t: Number of FIDs we compute for extrapolation
m_t: Precomputed groundtruth activations mean
c_t: Precomputed groundtruth activations covariance

procedure EVALUATE(G, I, n, t, m_t, c_t)
    FID = {}
    $z \leftarrow$ SOBOLSAMPLER(n)
    activations $\leftarrow I(G(z))$

    ▷ 5000 is the min no. of images we evaluate with
    batchSizes = LINSPACE($5000, n, t$)
    for batchSize in batchSizes **do**
        SHUFFLE(activations)
        activations$_i \leftarrow$ activations[:batchSize]
        FID.insert(CALCULATEFID(activations$_i$, m_t, c_t))
    end for
    reg $\leftarrow$ LINEARREGRESSION(1/batchSizes, FID)
    $\overline{\text{FID}}_\infty \leftarrow$ reg.predict(0)
    return $\overline{\text{FID}}_\infty$
end procedure

```python
from botorch.sampling.qmc import NormalQMCEngine

class randn_sampler():
    """
    Generates z~N(0,1) using random sampling or scrambled Sobol sequences.
    Args:
        ndim: (int)
            The dimension of z.
        use_sobol: (bool)
            If True, sample z from scrambled Sobol sequence. Else, sample
            from standard normal distribution.
            Default: False
        use_inv: (bool)
            If True, use inverse CDF to transform z from U[0,1] to N(0,1).
            Else, use Box-Muller transformation.
            Default: True
        cache: (bool)
            If True, we cache some amount of Sobol points and reorder them.
            This is mainly used for training GANs when we use two separate
            Sobol generators which helps stabilize the training.
            Default: False

    Examples::

        >>> sampler = randn_sampler(128, True)
        >>> z = sampler.draw(10) # Generates [10, 128] vector
    """

    def __init__(self, ndim, use_sobol=False, use_inv=True, cache=False):
        self.ndim = ndim
        self.cache = cache
        if use_sobol:
            self.sampler = NormalQMCEngine(d=ndim, inv_transform=use_inv)
            self.cached_points = torch.tensor([])
        else:
            self.sampler = None

    def draw(self, batch_size):
        if self.sampler is None:
            return torch.randn([batch_size, self.ndim])
        else:
            if self.cache:
                if len(self.cached_points) < batch_size:
                    # sample from sampler and reorder the points
                    self.cached_points = self.sampler.draw(int(1e6))[torch.randperm(int(1e6))]

                # Sample without replacement from cached points
                samples = self.cached_points[:batch_size]
                self.cached_points = self.cached_points[batch_size:]
                return samples
            else:
                return self.sampler.draw(batch_size)
```

Listing 1: PyTorch code for generating Sobol points