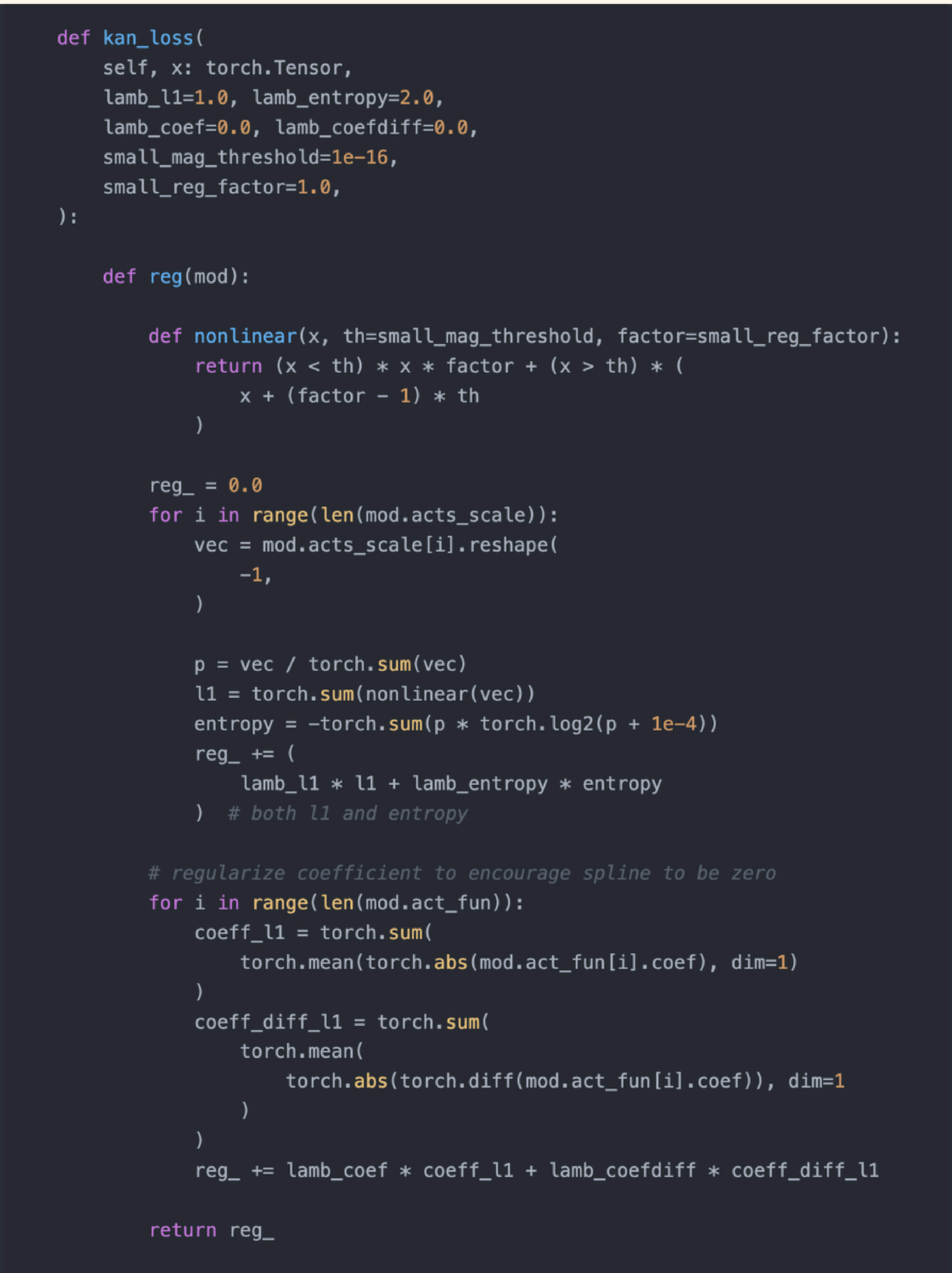
KAN GPT

GPT using Kolmogorov-Arnold Networks (KANs) for LM

1. KAN **loss function**

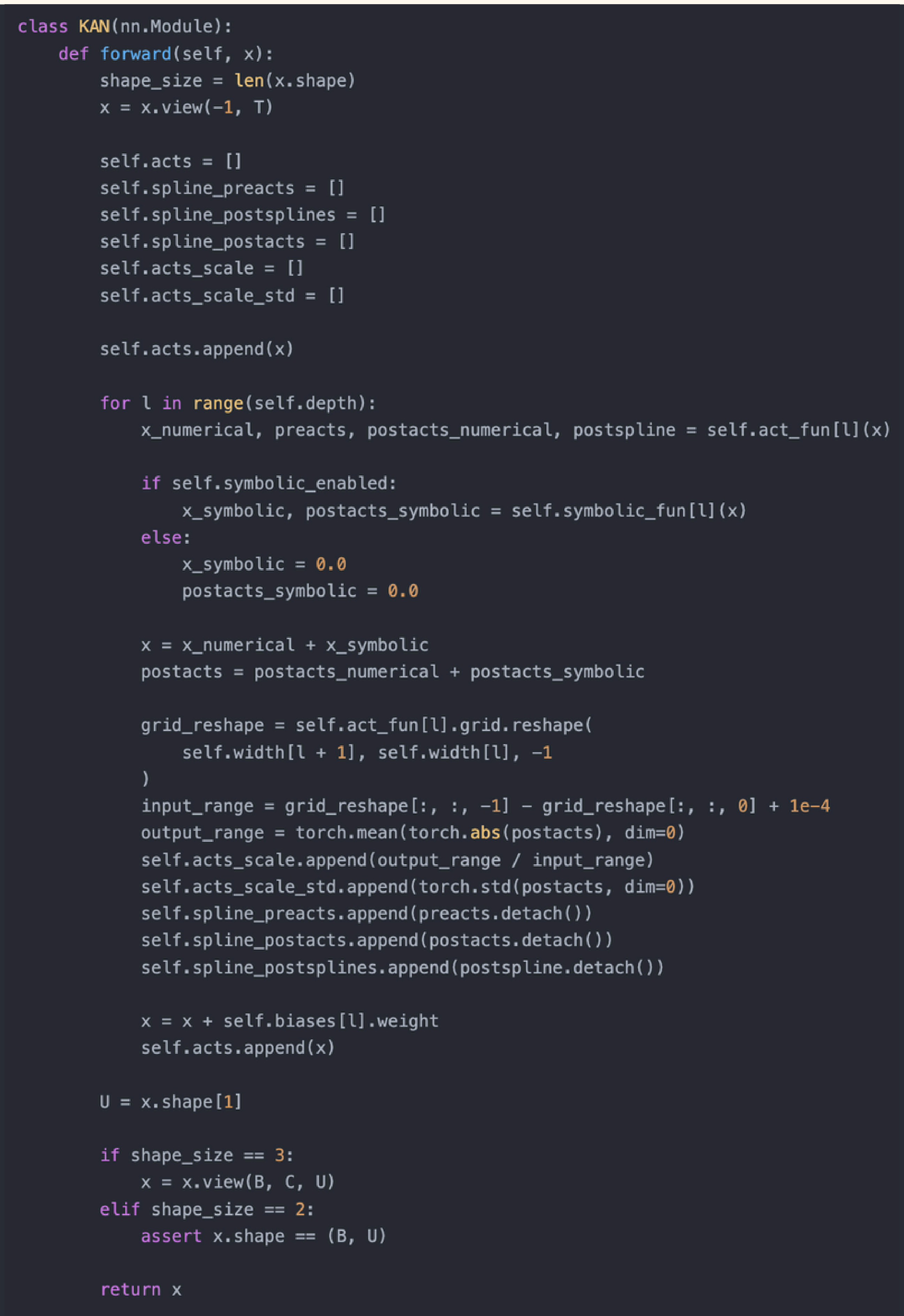


```python
def kan_loss(
    self, x: torch.Tensor,
    lamb_l1=1.0, lamb_entropy=2.0,
    lamb_coef=0.0, lamb_coefdiff=0.0,
    small_mag_threshold=1e-16,
    small_reg_factor=1.0,
):

    def reg(mod):

        def nonlinear(x, th=small_mag_threshold, factor=small_reg_factor):
            return (x < th) * x * factor + (x > th) * (
                x + (factor - 1) * th
            )

        reg_ = 0.0
        for i in range(len(mod.acts_scale)):
            vec = mod.acts_scale[i].reshape(
                -1,
            )

            p = vec / torch.sum(vec)
            l1 = torch.sum(nonlinear(vec))
            entropy = -torch.sum(p * torch.log2(p + 1e-4))
            reg_ += (
                lamb_l1 * l1 + lamb_entropy * entropy
            )  # both l1 and entropy

        # regularize coefficient to encourage spline to be zero
        for i in range(len(mod.act_fun)):
            coeff_l1 = torch.sum(
                torch.mean(torch.abs(mod.act_fun[i].coef), dim=1)
            )
            coeff_diff_l1 = torch.sum(
                torch.mean(
                    torch.abs(torch.diff(mod.act_fun[i].coef)), dim=1
                )
            )
            reg_ += lamb_coef * coeff_l1 + lamb_coefdiff * coeff_diff_l1

        return reg_
```

KAN GPT

GPT using Kolmogorov-Arnold Networks (KANs) for LM

2. KAN **forward pass**

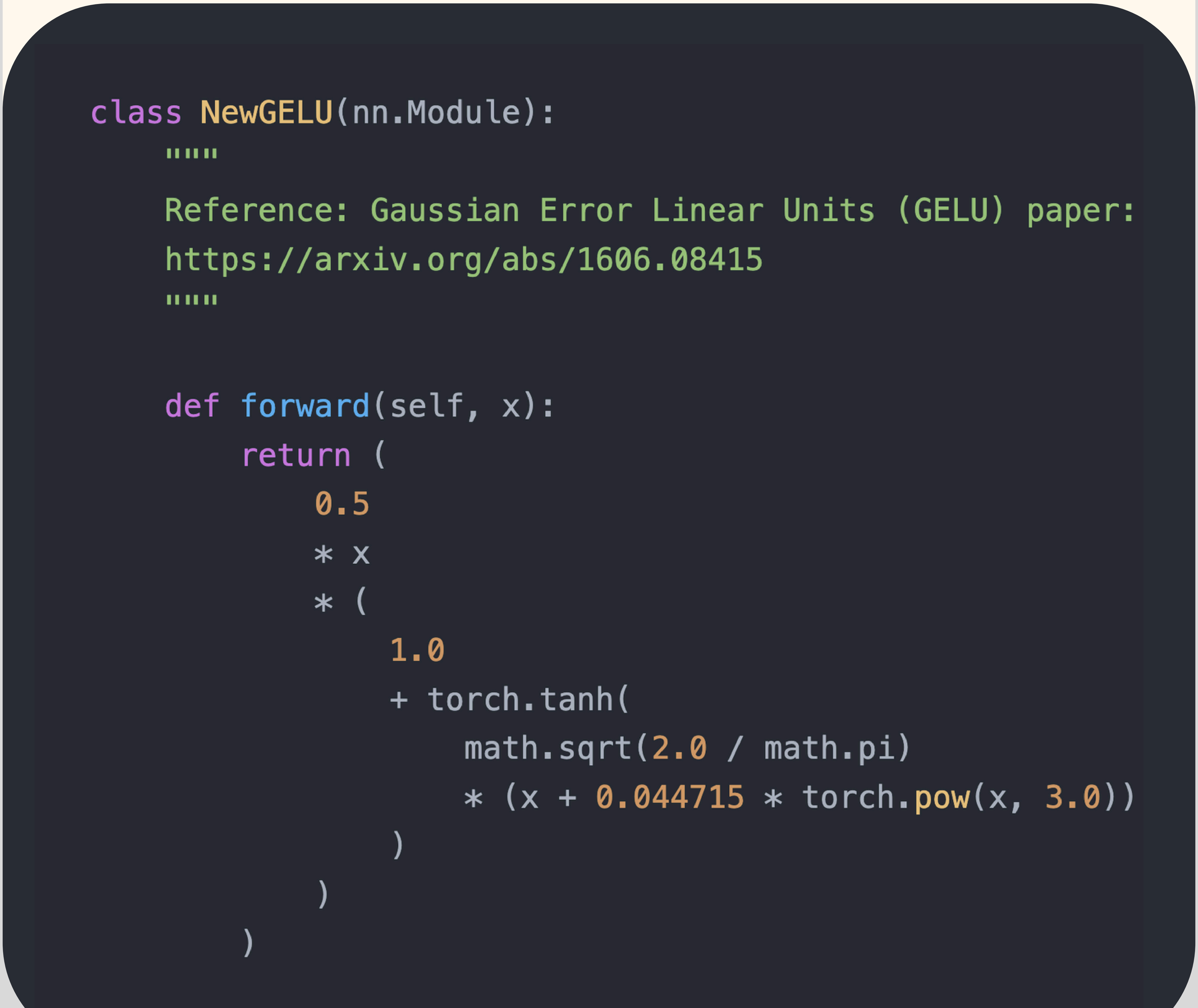


```python
class KAN(nn.Module):
    def forward(self, x):
        shape_size = len(x.shape)
        x = x.view(-1, T)

        self.acts = []
        self.spline_preacts = []
        self.spline_postsplines = []
        self.spline_postacts = []
        self.acts_scale = []
        self.acts_scale_std = []

        self.acts.append(x)

        for l in range(self.depth):
            x_numerical, preacts, postacts_numerical, postspline = self.act_fun[l](x)

            if self.symbolic_enabled:
                x_symbolic, postacts_symbolic = self.symbolic_fun[l](x)
            else:
                x_symbolic = 0.0
                postacts_symbolic = 0.0

            x = x_numerical + x_symbolic
            postacts = postacts_numerical + postacts_symbolic

            grid_reshape = self.act_fun[l].grid.reshape(
                self.width[l + 1], self.width[l], -1
            )
            input_range = grid_reshape[:, :, -1] - grid_reshape[:, :, 0] + 1e-4
            output_range = torch.mean(torch.abs(postacts), dim=0)
            self.acts_scale.append(output_range / input_range)
            self.acts_scale_std.append(torch.std(postacts, dim=0))
            self.spline_preacts.append(preacts.detach())
            self.spline_postacts.append(postacts.detach())
            self.spline_postsplines.append(postspline.detach())

            x = x + self.biases[l].weight
            self.acts.append(x)

        U = x.shape[1]

        if shape_size == 3:
            x = x.view(B, C, U)
        elif shape_size == 2:
            assert x.shape == (B, U)

        return x
```

KAN GPT

GPT using Kolmogorov-Arnold
Networks (KANs) for LM

3. GELU **Activation Function**



```python
class NewGELU(nn.Module):
    """
    Reference: Gaussian Error Linear Units (GELU) paper:
    https://arxiv.org/abs/1606.08415
    """

    def forward(self, x):
        return (
            0.5
            * x
            * (
                1.0
                + torch.tanh(
                    math.sqrt(2.0 / math.pi)
                    * (x + 0.044715 * torch.pow(x, 3.0))
                )
            )
        )
```