```
import torch
import torch.nn as nn
from loguru import logger
from zeta import SSM  # Ensure SSM is correctly imported
import torch.nn.functional as F


class HierarchicalMamba(nn.Module):
    """
    Hierarchical Mamba model composed of multiple Mamba2 blocks in a hierarchical structure.
    Each layer processes the input at different scales by progressively reducing the sequence length.

    Args:
        d_model (int): Dimensionality of the input and output features.
        num_layers (int): Number of Mamba2 blocks to stack hierarchically.
        d_state (int): Dimensionality of the expanded state.
        dt_rank (int): Rank of the discrete-time state-space model (SSM).
        dim_inner (int): Inner dimension used for state-space expansion.
    """

    def __init__(
        self,
        d_model: int,
        num_layers: int,
        dt_rank: int = 1,
        d_state: int = 64,
```

```
    d_conv: int = 4,
    expand: int = 2,
    dim_inner: int = None,
) -> None:
    super(HierarchicalMamba, self).__init__()

    self.num_layers = num_layers
    self.d_model = d_model

    # Default to d_model if dim_inner is None
    if dim_inner is None:
        dim_inner = d_model

    # Create a list of SSM layers
    self.layers = nn.ModuleList(
        [
            SSM(
                in_features=d_model,
                dt_rank=dt_rank,
                dim_inner=dim_inner,
                d_state=d_state,
            )
            for _ in range(num_layers)
        ]
    )
```

```
        logger.info(
                "HierarchicalMamba initialized with {} layers, d_model={}, d_state={}, dt_rank={},
dim_inner={}",
            num_layers,
            d_model,
            d_state,
            dt_rank,
            dim_inner,
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        """
        Forward pass through the Hierarchical Mamba model.

        Args:
            x (torch.Tensor): Input tensor of shape (batch_size, seq_length, d_model).

        Returns:
            torch.Tensor: Output tensor of shape (batch_size, seq_length // 2^num_layers, d_model).
        """
        logger.debug("HierarchicalMamba input shape: {}", x.shape)

        for i, layer in enumerate(self.layers):
            x = layer(x)
            logger.debug("After SSM layer {}, shape: {}", i, x.shape)
```

```
            if i < self.num_layers - 1:
                # Downsample the sequence length by a factor of 2 for hierarchical processing
                x = F.avg_pool1d(
                    x.transpose(1, 2), kernel_size=2
                ).transpose(1, 2)
                logger.debug(
                    "Downsampled shape after layer {}: {}", i, x.shape
                )

        logger.debug(
            "HierarchicalMamba final output shape: {}", x.shape
        )
        return x


def initialize_hierarchical_mamba(
    d_model: int,
    num_layers: int,
    d_state: int = 64,
    dt_rank: int = 1,
    dim_inner: int = None,
) -> HierarchicalMamba:
    """
    Utility function to initialize a Hierarchical Mamba model with logging.

    Args:
```

```
        d_model (int): Dimensionality of the input/output features.
        num_layers (int): Number of hierarchical layers.
        d_state (int, optional): Dimensionality of the expanded state (default=64).
        dt_rank (int, optional): Rank of the discrete-time state-space model (default=1).
        dim_inner (int, optional): Inner dimensionality for state-space (default=None).

    Returns:
        HierarchicalMamba: The initialized Hierarchical Mamba model.
    """
    logger.info(
        "Initializing Hierarchical Mamba model with d_model={}, num_layers={}, d_state={}, dt_rank={},
dim_inner={}",
        d_model,
        num_layers,
        d_state,
        dt_rank,
        dim_inner,
    )

    model = HierarchicalMamba(
        d_model,
        num_layers,
        dt_rank=dt_rank,
        d_state=d_state,
        dim_inner=dim_inner,
    ).to("cpu")
```

```
    logger.success(
        "Hierarchical Mamba model initialized successfully."
    )
    return model


# Example usage
if __name__ == "__main__":
    dim: int = 128  # Example model dimension
    num_layers: int = 3  # Number of hierarchical layers

    model = initialize_hierarchical_mamba(
        d_model=dim,
        num_layers=num_layers,
        d_state=64,
        dt_rank=1,
        dim_inner=64,  # Adjust inner dimension if necessary
    )

    # Sample input (batch_size, sequence_length, d_model)
    x = torch.randn(16, 100, dim).to("cpu")
    y = model(x)

    logger.info("Input shape: {}", x.shape)
    logger.info("Output shape: {}", y.shape)
```