A. Appendix

We present the PyTorch code for Hybrid Attention:

```python
def hybrid_attention(queries, keys, k, n):
    # Number of iterations
    t = (keys.shape[2] - n) // k

    # Step 1: Dense Attention
    prompt_keys = keys[:, :, :n, :]
    prompt_attn = torch.matmul(queries, prompt_keys.transpose(2, 3))

    # Step 2: Sparse Attention
    gen_keys = keys[:, :, n:, :]
    reshaped_keys = gen_keys.reshape(
        (keys.shape[0], keys.shape[1],t,k, keys.shape[3])).transpose(2,3)
    element_wise = torch.einsum('abcx, abcdx -> abcd', queries, reshaped_keys)
    mask = torch.eye(k). repeat(1, t)[None, None,:,:].repeat(
        keys.shape[0], keys.shape[1], 1, 1).bool().to(element_wise.get_device())
    gen_mask = mask.clone().to(dtype=element_wise.dtype)
    gen_mask[mask] = element_wise.flatten()
    min_dtype = torch.finfo(gen_mask.dtype).min
    gen_mask[~mask] = min_dtype
    attn_weights = torch.cat([prompt_attn, gen_mask], dim=-1)

    return attn_weights
```

Our approach without Hybrid Attention:

```python
def naive_multisequence_attention(queries, keys, k, n):

    t = (keys.shape[2] - n) // k
    attn_weights = torch.matmul(queries, keys.transpose(2, 3))

    prompt_mask = torch.ones((k, n))
    continuation_mask = torch.eye(k).repeat(1, t)
    mask = torch.cat([prompt_mask, continuation_mask], dim=-1)[None, None, :,:].repeat(
        1, attn_weights.shape[1], 1, 1).bool()
    min_dtype = torch.finfo(attn_weights.dtype).min
    attn_weights[~mask] = min_dtype

    return attn_weights
```