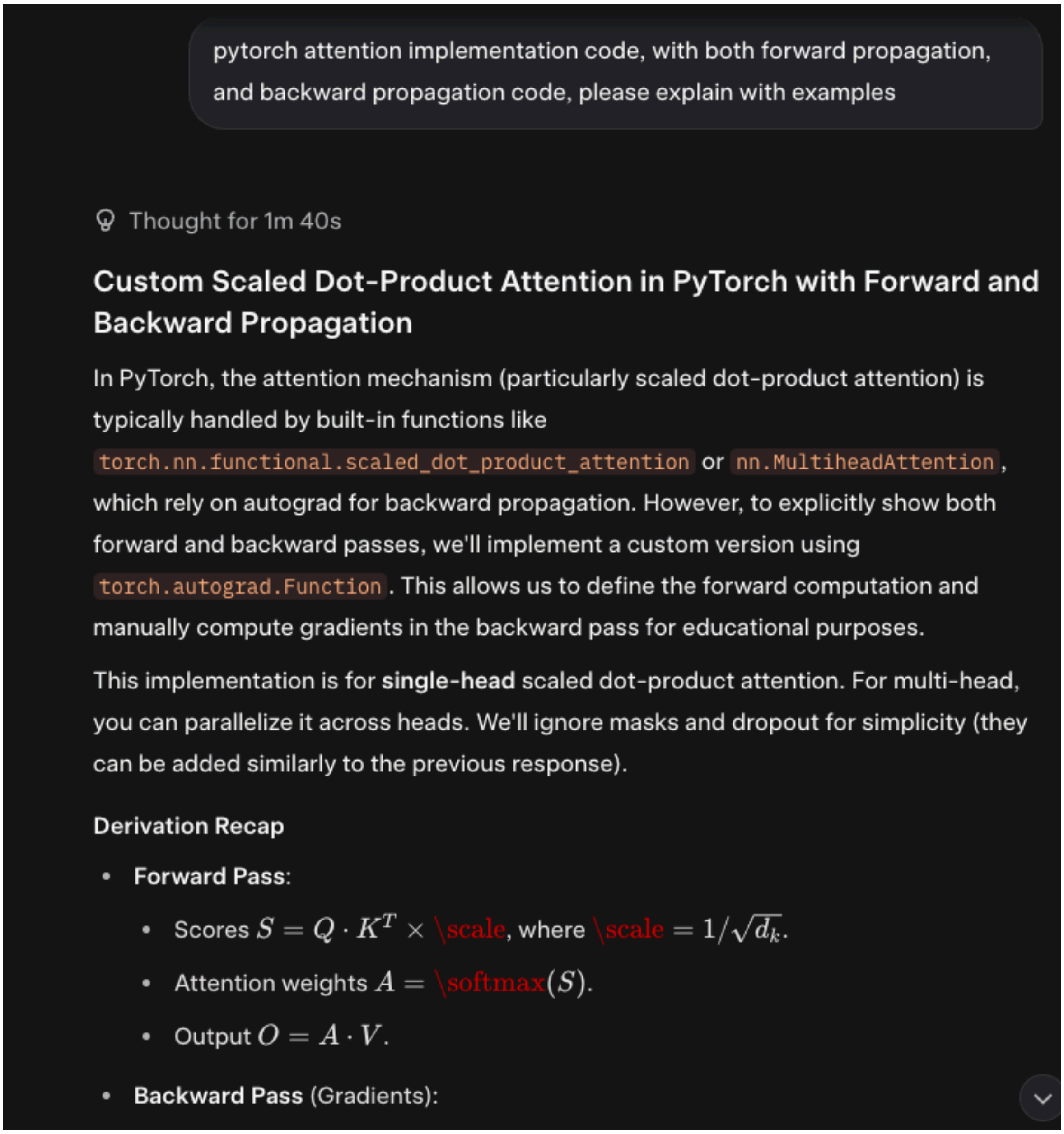


pytorch attention implementation code, with both forward propagation, and backward propagation code, please explain with examples

Thought for 1m 40s

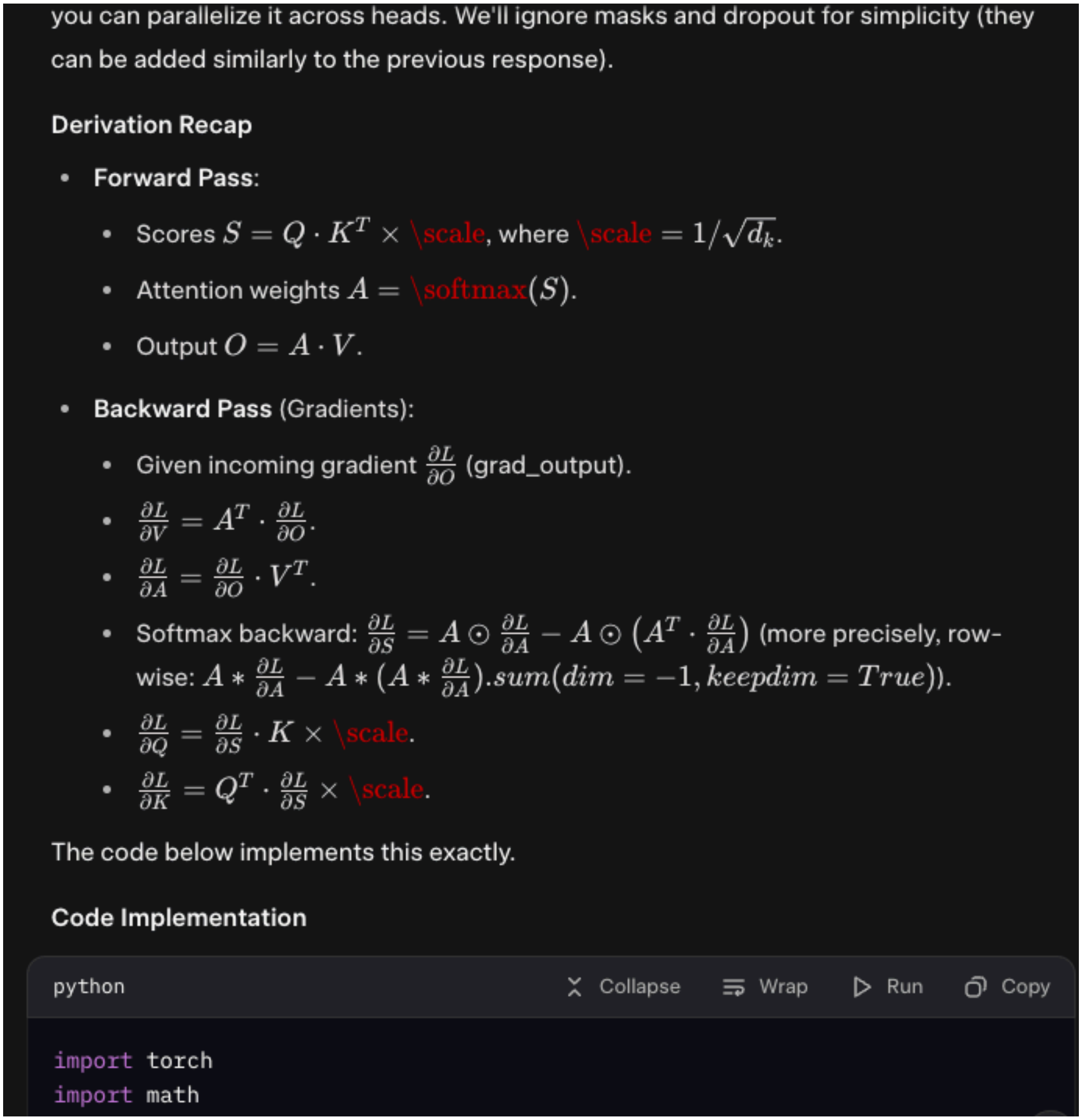
Custom Scaled Dot-Product Attention in PyTorch with Forward and Backward Propagation

In PyTorch, the attention mechanism (particularly scaled dot-product attention) is typically handled by built-in functions like `torch.nn.functional.scaled_dot_product_attention` or `nn.MultiheadAttention`, which rely on autograd for backward propagation. However, to explicitly show both forward and backward passes, we'll implement a custom version using `torch.autograd.Function`. This allows us to define the forward computation and manually compute gradients in the backward pass for educational purposes.

This implementation is for **single-head** scaled dot-product attention. For multi-head, you can parallelize it across heads. We'll ignore masks and dropout for simplicity (they can be added similarly to the previous response).

Derivation Recap

- **Forward Pass**:
  - Scores $S = Q \cdot K^T \times \scale$, where $\scale = 1/\sqrt{d_k}$.
  - Attention weights $A = \softmax(S)$.
  - Output $O = A \cdot V$.
- **Backward Pass** (Gradients):



you can parallelize it across heads. We'll ignore masks and dropout for simplicity (they can be added similarly to the previous response).

Derivation Recap

- **Forward Pass**:
    - Scores $S = Q \cdot K^T \times$ \scale, where \scale $= 1/\sqrt{d_k}$.
    - Attention weights $A =$ \softmax$(S)$.
    - Output $O = A \cdot V$.
- **Backward Pass** (Gradients):
    - Given incoming gradient $\frac{\partial L}{\partial O}$ (grad_output).
    - $\frac{\partial L}{\partial V} = A^T \cdot \frac{\partial L}{\partial O}$.
    - $\frac{\partial L}{\partial A} = \frac{\partial L}{\partial O} \cdot V^T$.
    - Softmax backward: $\frac{\partial L}{\partial S} = A \odot \frac{\partial L}{\partial A} - A \odot \left(A^T \cdot \frac{\partial L}{\partial A}\right)$ (more precisely, row-wise: $A * \frac{\partial L}{\partial A} - A * (A * \frac{\partial L}{\partial A}).sum(dim = -1, keepdim = True)$).
    - $\frac{\partial L}{\partial Q} = \frac{\partial L}{\partial S} \cdot K \times$ \scale.
    - $\frac{\partial L}{\partial K} = Q^T \cdot \frac{\partial L}{\partial S} \times$ \scale.

The code below implements this exactly.

Code Implementation

```python
import torch
import math
```