

BitNet

Content

I • Chapter 1
BitNet?

II • Chapter 2
BitNet 추론 방법

III • Chapter 3
추론 결과

BitNet?

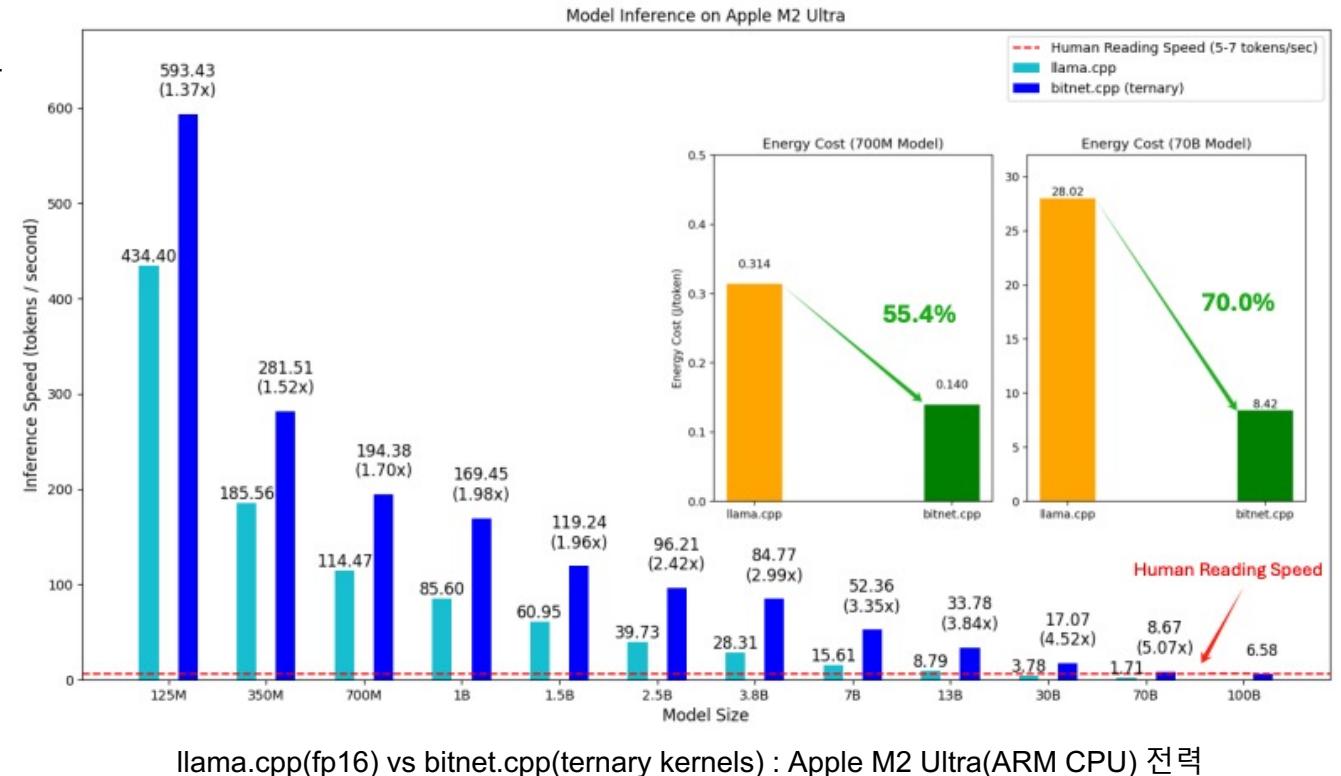
BitNet은 (-1,0,1) 값으로 가중치 파라미터를 양자화한 모델

기존 문제점:

- LLM 크기와 복잡성 커지면서 전력, 메모리 사용량이 큼

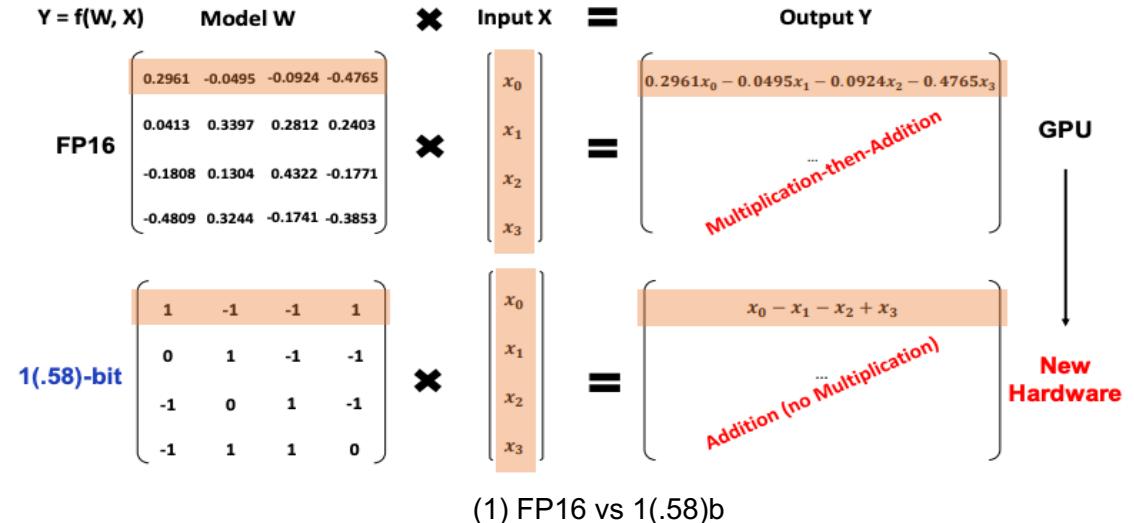
해결 방법:

- (-1,0,1) 3개의 값으로 양자화한 LLM 모델
- **BitNet b1.58 Model** $\log_2 3 = 1.5849..$

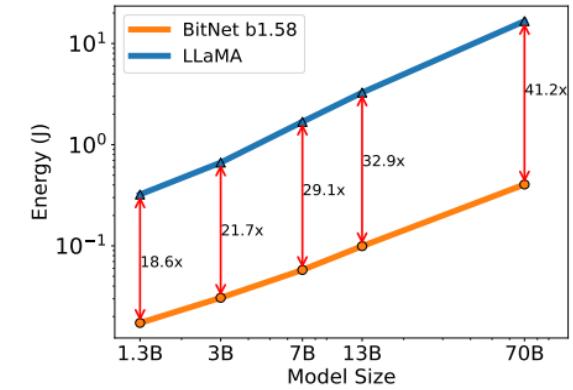
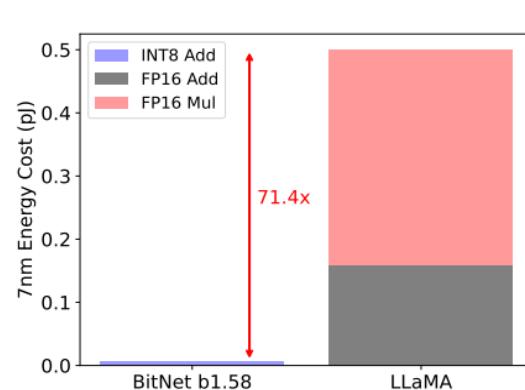


BitNet?

BitNet은 Llama 모델에 비해 행렬 곱셈에 사용되는 산술 연산 에너지를 **71.4배 절약**



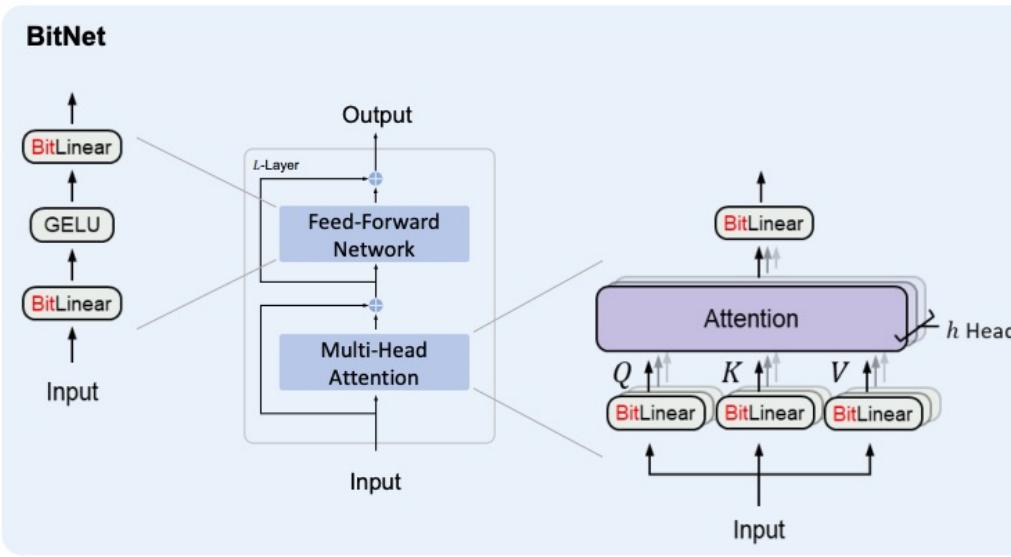
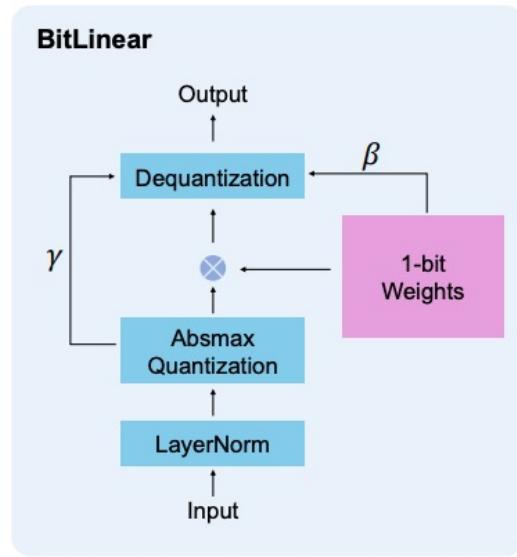
가중치 양자화 과정을 통해 곱셈 연산 대신 더하기 연산으로 가능



(2) LLaMA vs 1(1.58)b

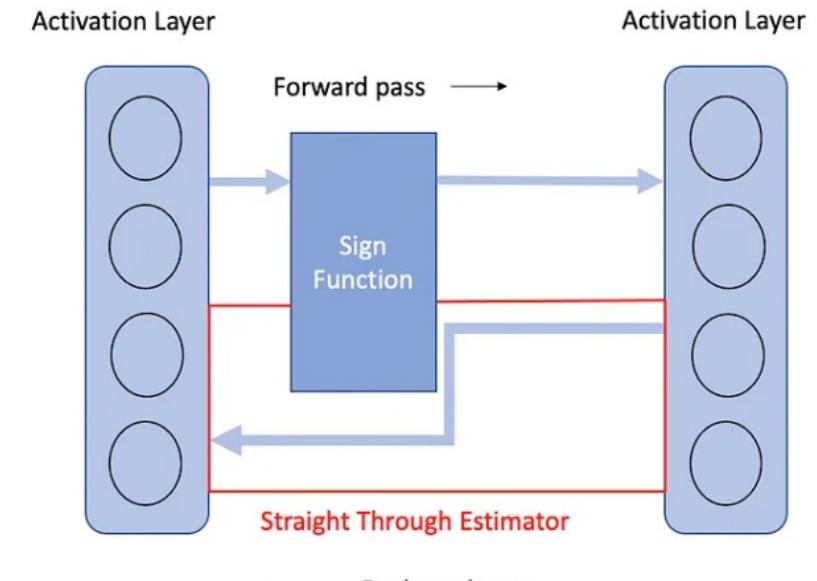
BitLinear, STE(Straight Through Estimator)

nn.Linear 대신 BitLinear을 사용, 역전파 시 양자화 이전 연산 그대로 사용



추론 과정 BitLinear 구조

BitLinear

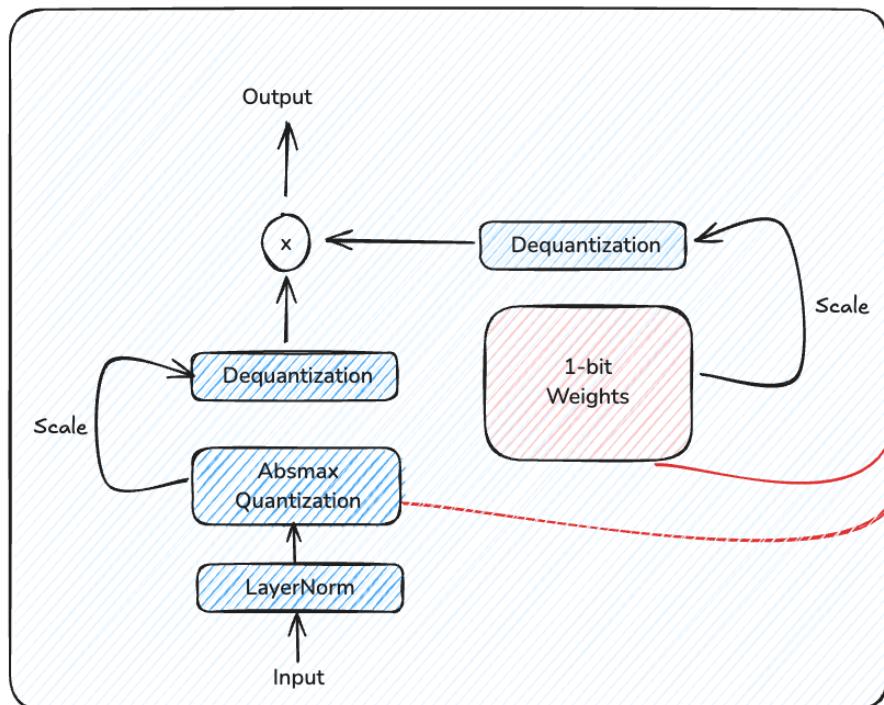


STE

BitLinear

$$\text{BitLinear} = \mathbf{Wx} = (\text{Weigth_quant}) \times (\text{입력텐서_quant})$$

Train



```
def weight_quant(weight, num_bits=1):
    dtype = weight.dtype
    weight = weight.float()
    s = 1 / weight.abs().mean().clamp(min=1e-5)
    result = (weight * s).round().clamp(-1, 1) / s
    return result.type(dtype)

def activation_quant(x, num_bits=8):
    dtype = x.dtype
    x = x.float()
    Qn = -2 ** (num_bits - 1)
    Qp = 2 ** (num_bits - 1) - 1
    s = Qp / x.abs().max(dim=-1, keepdim=True).values.clamp(min=1e-5)
    result = (x * s).round().clamp(Qn, Qp) / s
    return result.type(dtype)

class BitLinear(nn.Linear):
    def __init__(self, *kargs, weight_bits=1, input_bits=8, **kwargs):
        super(BitLinear, self).__init__(*kargs, **kwargs)
        """
        RMSNorm is placed outside BitLinear
        """
        self.weight_bits = weight_bits
        self.input_bits = input_bits

    def forward(self, input):
        quant_input = input + (activation_quant(input, self.input_bits) - input).detach()
        quant_weight = self.weight + (weight_quant(self.weight, self.weight_bits) - self.weight).detach()

        out = nn.functional.linear(quant_input, quant_weight)
        if not self.bias is None:
            out += self.bias.view(1, -1).expand_as(out)

        return out
```

Weight Matrix Quantization

Weight Ternary Quantization

BitNet b1.58 (This Work)

{-1, 0, 1}

$$W = \begin{pmatrix} 1 & -1 & \dots & 1 \\ 0 & \dots & -1 & -1 \\ -1 & 1 & \dots & 0 \\ -1 & \dots & 0 & -1 \end{pmatrix}$$



Transformer LLMs

16-bit Float (FP16/BF16)

$$W = \begin{pmatrix} 0.2961 & -0.0495 & \dots & -0.4765 \\ 0.0413 & \dots & 0.2812 & 0.2403 \\ -0.1808 & 0.1304 & \dots & -0.1771 \\ -0.4809 & \dots & -0.1741 & -0.3853 \end{pmatrix}$$

Weight Matrix Quantization

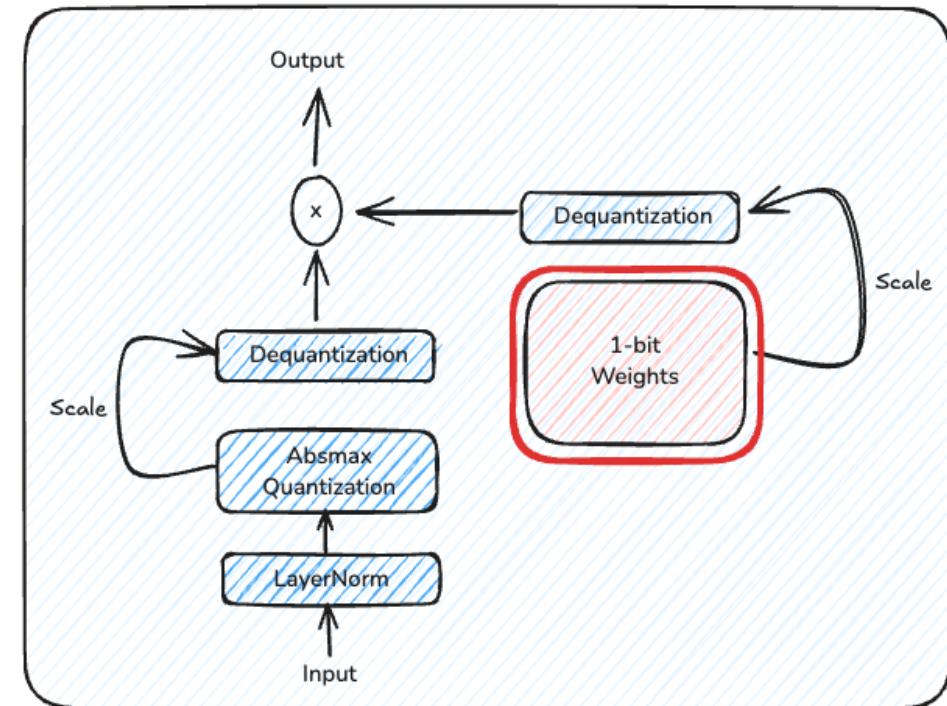
Weight 스케일 과정

```
def weight_quant(weight, num_bits=1):
    dtype = weight.dtype
    weight = weight.float()
    s = 1 / weight.abs().mean().clamp(min=1e-5)
    result = (weight * s).round().clamp(-1, 1) / s
    return result.type(dtype)
```

$$W = \begin{pmatrix} 0.8 & -0.5 & 1.2 \\ -1.5 & 0.4 & -0.9 \\ 1.3 & -0.7 & 0.2 \end{pmatrix}$$

$$\frac{1}{9}(0.8+0.5+1.2+1.5+0.4+0.9+1.3+0.7+0.2)$$

$$\gamma = \frac{1}{9}(7.5) = 0.8333$$



$$\widetilde{W} = \text{RoundClip}\left(\frac{W}{\gamma + \epsilon}, -1, 1\right),$$

$\text{RoundClip}(x, a, b) = \max(a, \min(b, \text{round}(x)))$,

$$\gamma = \frac{1}{nm} \sum_{ij} |W_{ij}|.$$

Weight Matrix Quantization

Weight에서 양자화 과정

```
def weight_quant(weight, num_bits=1):
    dtype = weight.dtype
    weight = weight.float()
    s = 1 / weight.abs().mean().clamp(min=1e-5)
    result = weight * s).round().clamp(-1, 1) / s
    return result.type(dtype)
```

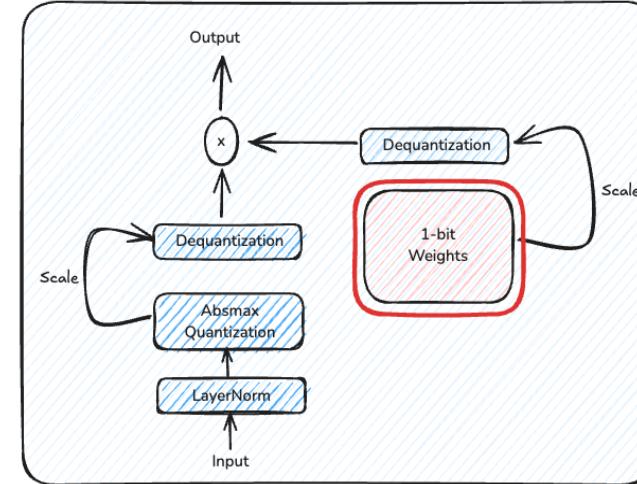
$$W = \begin{bmatrix} 0.8 & -0.5 & 1.2 \\ -1.5 & 0.4 & -0.9 \\ 1.3 & -0.7 & 0.2 \end{bmatrix}$$

$$\frac{1}{\gamma} = \frac{1}{0.8333} \approx 1.2$$

$$W \times \text{scale}_w = \begin{bmatrix} 0.8 \times 1.2 & -0.5 \times 1.2 & 1.2 \times 1.2 \\ -1.5 \times 1.2 & 0.4 \times 1.2 & -0.9 \times 1.2 \\ 1.3 \times 1.2 & -0.7 \times 1.2 & 0.2 \times 1.2 \end{bmatrix}$$



$$\begin{bmatrix} 0.96 & -0.6 & 1.44 \\ -1.8 & 0.48 & -1.08 \\ 1.56 & -0.84 & 0.24 \end{bmatrix}$$



$$\widetilde{W} = \text{RoundClip}\left(\frac{W}{\gamma + \epsilon}, -1, 1\right),$$

$\text{RoundClip}(x, a, b) = \max(a, \min(b, \text{round}(x)))$,

$$\gamma = \frac{1}{nm} \sum_{ij} |W_{ij}|.$$

Weight Matrix Quantization

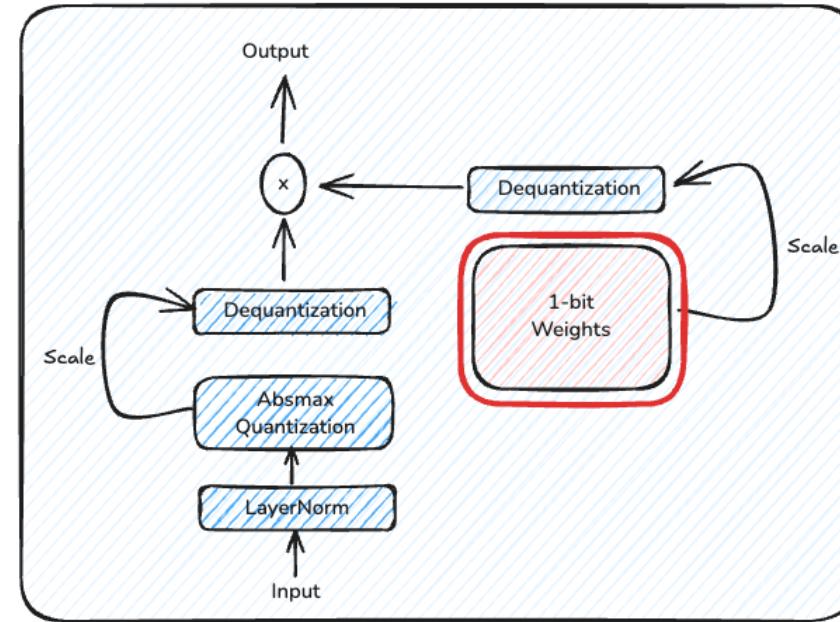
Weight에서 양자화 과정

```
def weight_quant(weight, num_bits=1):
    dtype = weight.dtype
    weight = weight.float()
    s = 1 / weight.abs().mean().clamp(min=1e-5)
    result = (weight * s).round().clamp(-1, 1) / s
    return result.type(dtype)
```

$$\begin{bmatrix} 0.96 & -0.6 & 1.44 \\ -1.8 & 0.48 & -1.08 \\ 1.56 & -0.84 & 0.24 \end{bmatrix}$$

$$\text{Weight_quant} = \begin{bmatrix} 1 & -1 & 1 \\ -1 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix}$$

$\Rightarrow \text{Max}(-1, \text{Min}(1, \text{round}(0.96)))$



$$\widetilde{W} = \text{RoundClip}\left(\frac{W}{\gamma + \epsilon}, -1, 1\right),$$

$$\text{RoundClip}(x, a, b) = \max(a, \min(b, \text{round}(x))),$$

$$\gamma = \frac{1}{nm} \sum_{ij} |W_{ij}|.$$

Weight Matrix Quantization

Dequantize Weight

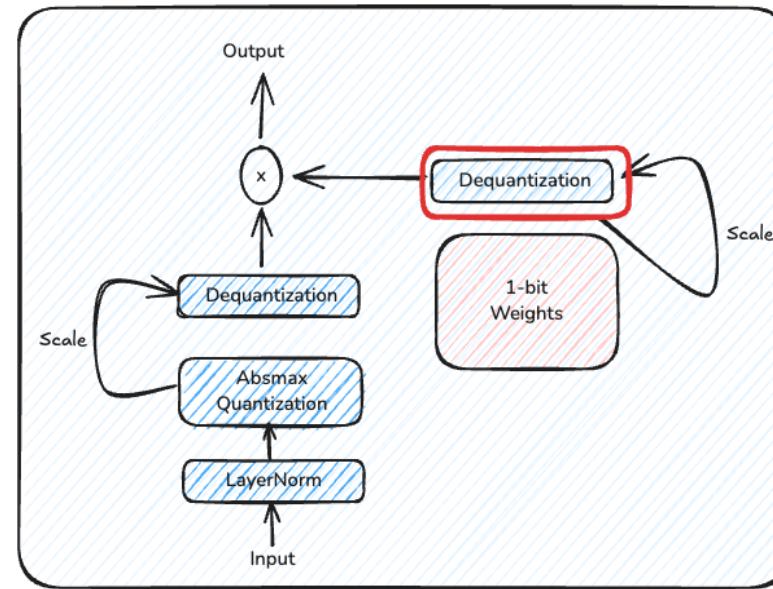
```
def weight_quant(weight, num_bits=1):
    dtype = weight.dtype
    weight = weight.float()
    s = 1 / weight.abs().mean().clamp(min=1e-5)
    result = (weight * s).round().clamp(-1, 1) / s
    return result.type(dtype)
```

Dequantize

$$\begin{bmatrix} 1 \times 0.8333 & -1 \times 0.8333 & 1 \times 0.8333 \\ -1 \times 0.8333 & 0 \times 0.8333 & -1 \times 0.8333 \\ 1 \times 0.8333 & -1 \times 0.8333 & 0 \times 0.8333 \end{bmatrix}$$

↓

$$\begin{bmatrix} 0.8333 & -0.8333 & 0.8333 \\ -0.8333 & 0 & -0.8333 \\ 0.8333 & -0.8333 & 0 \end{bmatrix}$$



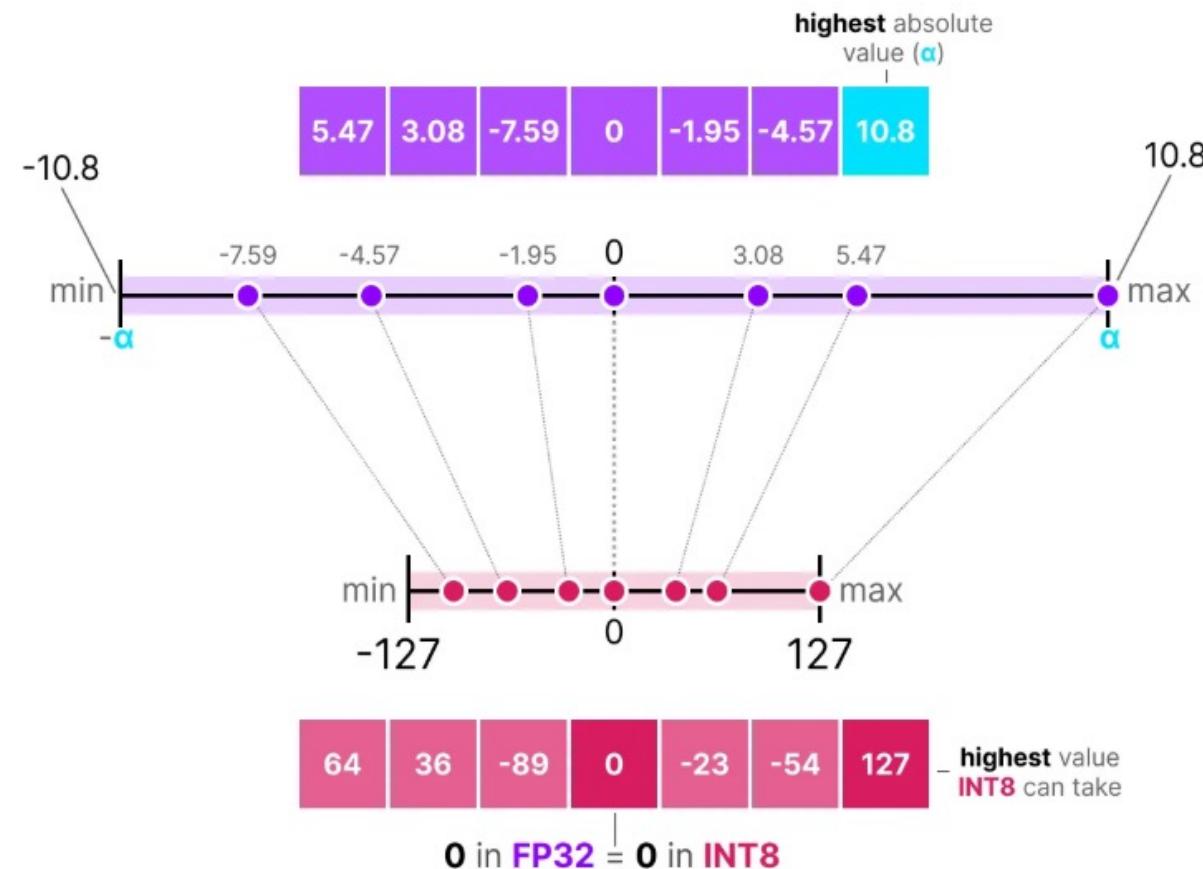
$$\widetilde{W} = \text{RoundClip}\left(\frac{W}{\gamma + \epsilon}, -1, 1\right),$$

$$\text{RoundClip}(x, a, b) = \max(a, \min(b, \text{round}(x))),$$

$$\gamma = \frac{1}{nm} \sum_{ij} |W_{ij}|.$$

Activation Matrix Quantization

Absmax 양자화



Activation Matrix Quantization

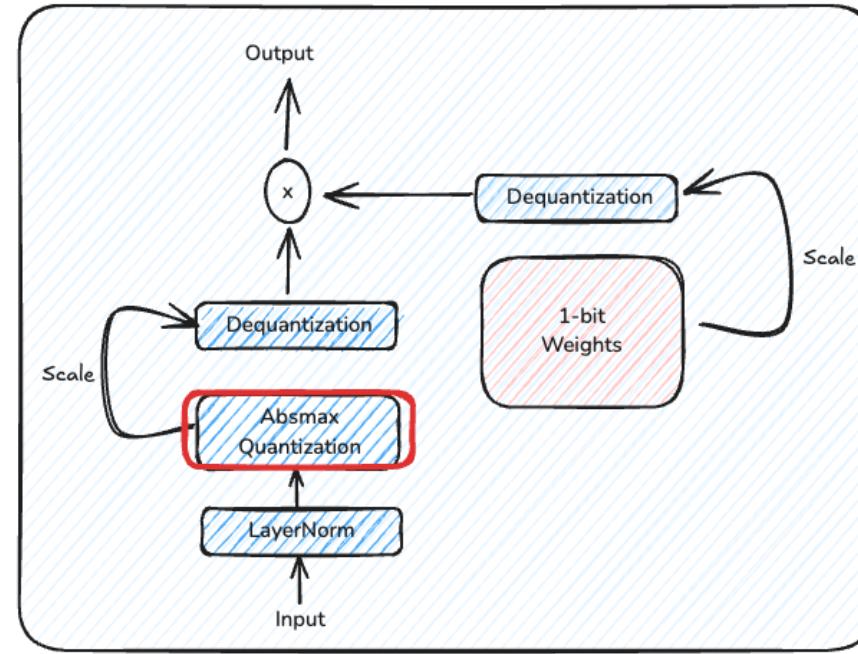
$[-Q_b, Q_b]$ 범위

```
def activation_quant(x, num_bits=8):
    dtype = x.dtype
    x = x.float()
    Qn = -2 ** (num_bits - 1)
    Qp = 2 ** (num_bits - 1) - 1
    s = Qp / x.abs().max(dim=-1, keepdim=True).values.clamp(min=1e-5)
    result = (x * s).round().clamp(Qn, Qp) / s
    return result.type(dtype)
```

INT8의 범위

$$Q_n = -128$$

$$Q_p = 127$$



$$\tilde{x} = \text{Quant}(x) = \text{Clip}\left(x \times \frac{Q_b}{\gamma}, -Q_b + \epsilon, Q_b - \epsilon\right),$$

$$\text{Clip}(x, a, b) = \max(a, \min(b, x)), \quad \gamma = \|x\|_\infty,$$

$$[-Q_b, Q_b] (Q_b = 2^{b-1})$$

Activation Matrix Quantization

Absmax

```
def activation_quant(x, num_bits=8):
    dtype = x.dtype
    x = x.float()
    Qn = -2 ** (num_bits - 1)
    Qp = 2 ** (num_bits - 1) - 1
    s = Qp / x.abs().max(dim=-1, keepdim=True).values.clamp(min=1e-5)
    result = (x * s).round().clamp(Qn, Qp) / s
    return result.type(dtype)
```

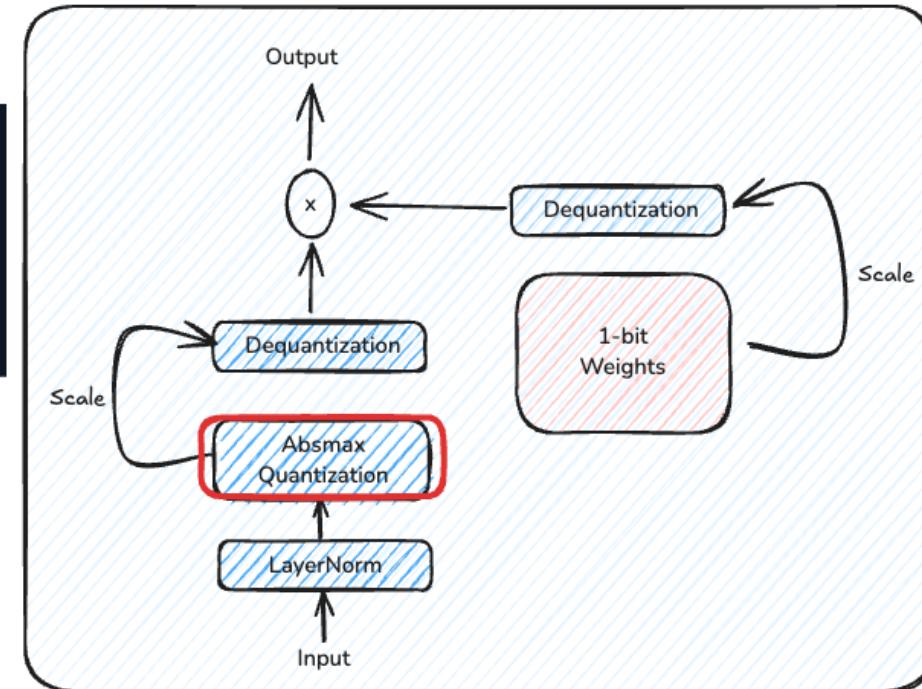
$$x = \begin{bmatrix} 1.0 & -0.6 & 0.7 \\ -0.9 & 0.4 & -1.2 \\ 0.8 & -0.5 & 0.3 \end{bmatrix}$$

↓

Row1 : 최대 절대값 = 1.0

Row2 : 최대 절대값 = 1.2

Row3 : 최대 절대값 = 0.8



$$\tilde{x} = \text{Quant}(x) = \text{Clip}\left(x \times \frac{Q_b}{\gamma}, -Q_b + \epsilon, Q_b - \epsilon\right),$$

$$\text{Clip}(x, a, b) = \max(a, \min(b, x)), \quad \gamma = \|x\|_\infty,$$

$$[-Q_b, Q_b] \quad (Q_b = 2^{b-1})$$

Activation Matrix Quantization

Scale 계산

```
def activation_quant(x, num_bits=8):
    dtype = x.dtype
    x = x.float()
    Qn = -2 ** (num_bits - 1)
    Qp = 2 ** (num_bits - 1) - 1
    s = Qp / x.abs().max(dim=-1, keepdim=True).values.clamp(min=1e-5)
    result = (x * s).round().clamp(Qn, Qp) / s
    return result.type(dtype)
```

Row1 : 최대 절대값 = 1.0

Row2 : 최대 절대값 = 1.2

Row3 : 최대 절대값 = 0.8

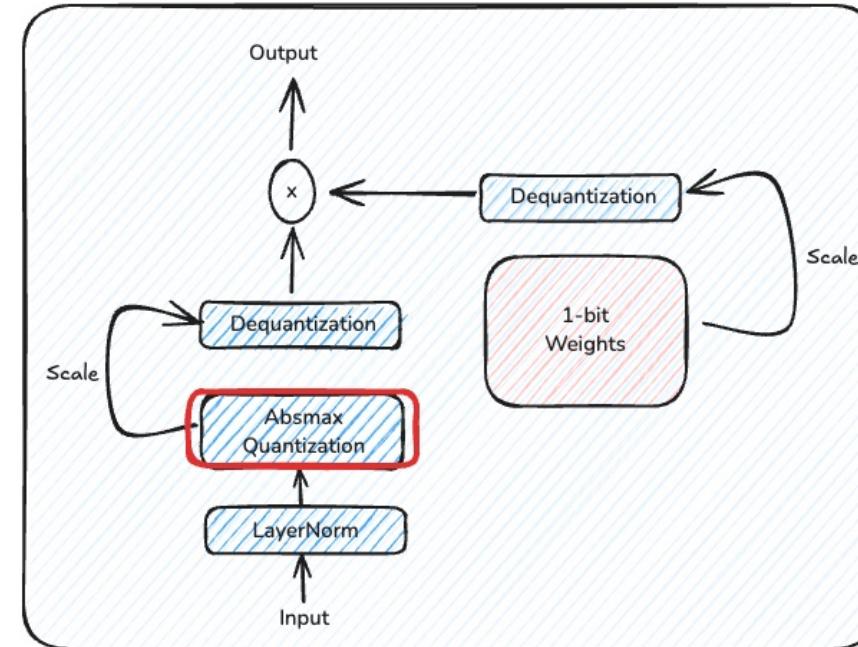
$$\text{scale} = \begin{pmatrix} \frac{127}{1.0} \\ \frac{127}{1.2} \\ \frac{127}{0.8} \end{pmatrix} = \begin{pmatrix} 127 \\ 105.83 \\ 158.75 \end{pmatrix}$$

$$\tilde{x} = \text{Quant}(x) = \text{Clip}\left(x \times \frac{Q_b}{\gamma}, -Q_b + \epsilon, Q_b - \epsilon\right),$$

$$\text{Clip}(x, a, b) = \max(a, \min(b, x)), \quad \gamma = \|x\|_\infty,$$

$$[-Q_b, Q_b] \quad (Q_b = 2^{b-1})$$

$$Q_n = -128, Q_p = 127$$



Activation Matrix Quantization

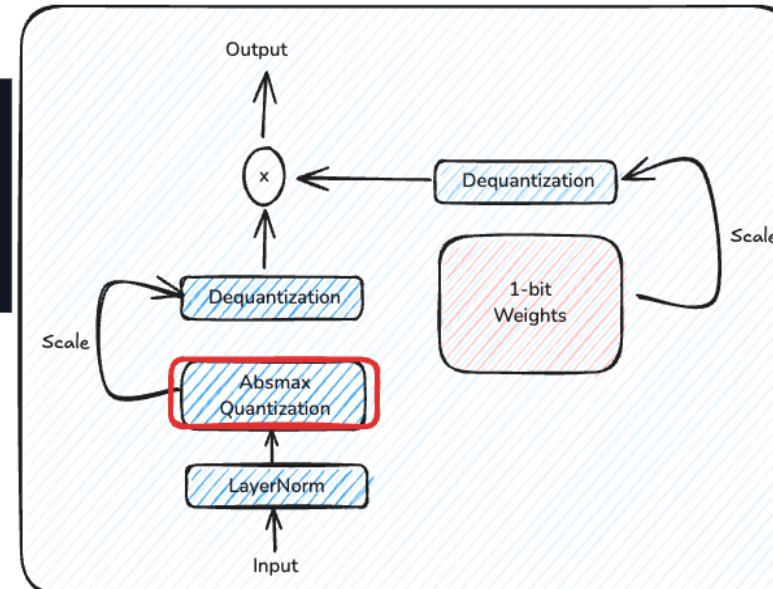
Absmax

```
def activation_quant(x, num_bits=8):
    dtype = x.dtype
    x = x.float()
    Qn = -2 ** (num_bits - 1)
    Qp = 2 ** (num_bits - 1) - 1
    s = Qp / x.abs().max(dim=-1, keepdim=True).values.clamp(min=1e-5)
    result = (x * s).round().clamp(Qn, Qp) / s
    return result.type(dtype)
```

$$x = \begin{bmatrix} 1.0 & -0.6 & 0.7 \\ -0.9 & 0.4 & -1.2 \\ 0.8 & -0.5 & 0.3 \end{bmatrix} \quad \text{scale} = \begin{bmatrix} 127 \\ 105.83 \\ 158.75 \end{bmatrix}$$

$$x \times \text{scale} = \begin{bmatrix} 1.0 \times 127 & -0.6 \times 127 & 0.7 \times 127 \\ -0.9 \times 105.83 & 0.4 \times 105.83 & -1.2 \times 105.83 \\ 0.8 \times 158.75 & -0.5 \times 158.75 & 0.3 \times 158.75 \end{bmatrix}$$

$$\begin{bmatrix} 127 & -76.2 & 88.9 \\ -95.2 & 42.3 & -127 \\ 127 & -79.4 & 47.6 \end{bmatrix} \xrightarrow{\text{Round}} \begin{bmatrix} 127 & -76 & 89 \\ -95 & 42 & -127 \\ 127 & -79 & 48 \end{bmatrix}$$



$$\tilde{x} = \text{Quant}(x) = \text{Clip}\left(x \times \frac{Q_b}{\gamma}, -Q_b + \epsilon, Q_b - \epsilon\right),$$

$$\text{Clip}(x, a, b) = \max(a, \min(b, x)), \quad \gamma = \|x\|_\infty,$$

$$[-Q_b, Q_b] (Q_b = 2^{b-1})$$

$$Q_n = -128, Q_p = 127$$

Activation Matrix Quantization

Absmax 양자화

```
def activation_quant(x, num_bits=8):
    dtype = x.dtype
    x = x.float()
    Qn = -2 ** (num_bits - 1)
    Qp = 2 ** (num_bits - 1) - 1
    s = Qp / x.abs().max(dim=-1, keepdim=True).values.clamp(min=1e-5)
    result = (x * s).round().clamp(Qn, Qp) / s
    return result.type(dtype)
```

$$x \times \text{scale} = \begin{bmatrix} 127 & -76 & 89 \\ -95 & 42 & -127 \\ 127 & -79 & 48 \end{bmatrix}$$

$$\text{Absmax} = \begin{bmatrix} 127 & \textcircled{-76} & 89 \\ -95 & 42 & -127 \\ 127 & -79 & 48 \end{bmatrix}$$

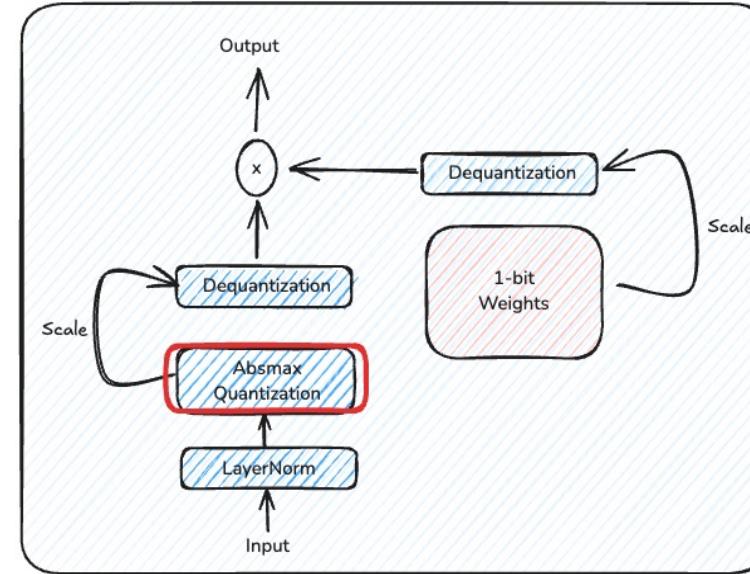
Max(-128, Min(127, -76))

$$\tilde{x} = \text{Quant}(x) = \text{Clip}\left(x \times \frac{Q_b}{\gamma}, -Q_b + \epsilon, Q_b - \epsilon\right),$$

$$\text{Clip}(x, a, b) = \max(a, \min(b, x)), \quad \gamma = \|x\|_\infty,$$

$$[-Q_b, Q_b] \quad (Q_b = 2^{b-1})$$

$$Q_n = -128, Q_p = 127$$



Activation Matrix Quantization

Dequantize Activation Matrix

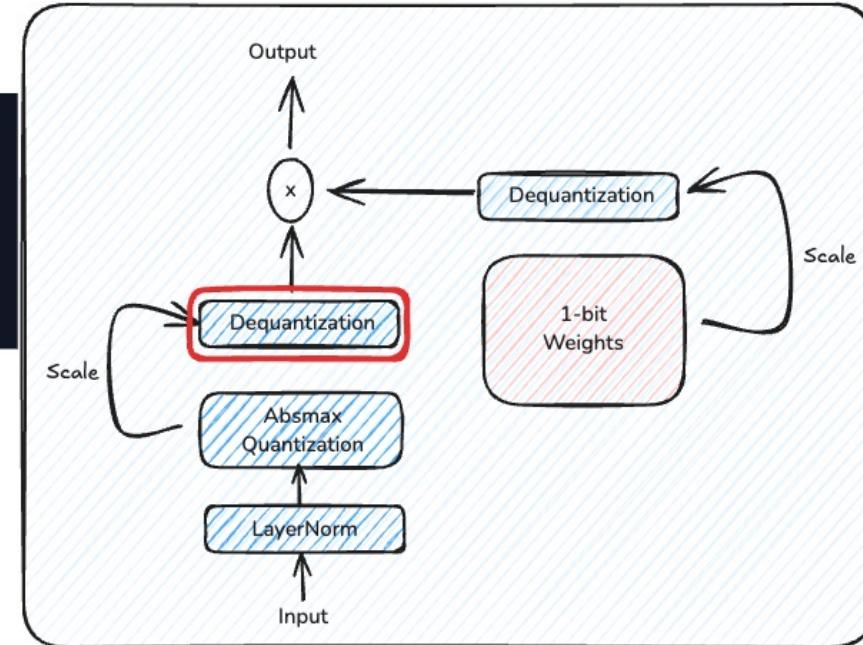
```
def activation_quant(x, num_bits=8):
    dtype = x.dtype
    x = x.float()
    Qn = -2 ** (num_bits - 1)
    Qp = 2 ** (num_bits - 1) - 1
    s = Qp / x.abs().max(dim=-1, keepdim=True).values.clamp(min=1e-5)
    result = (x * s).round().clamp(Qn, Qp) / s
    return result.type(dtype)
```

$$Absmax = \begin{bmatrix} 127 & -76 & 89 \\ -95 & 42 & -127 \\ 127 & -79 & 48 \end{bmatrix} \quad scale = \begin{bmatrix} 127 \\ 105.83 \\ 158.75 \end{bmatrix}$$

$$\text{Dequantize } (Absmax \times 1/s) = \begin{bmatrix} 1.0 & -0.6 & 0.7 \\ -0.9 & 0.4 & -1.2 \\ 0.8 & -0.5 & 0.3 \end{bmatrix}$$

$$\tilde{x} = \text{Quant}(x) = \text{Clip}\left(x \times \frac{Q_b}{\gamma}, -Q_b + \epsilon, Q_b - \epsilon\right),$$

$$\text{Clip}(x, a, b) = \max(a, \min(b, x)), \quad \gamma = \|x\|_\infty,$$

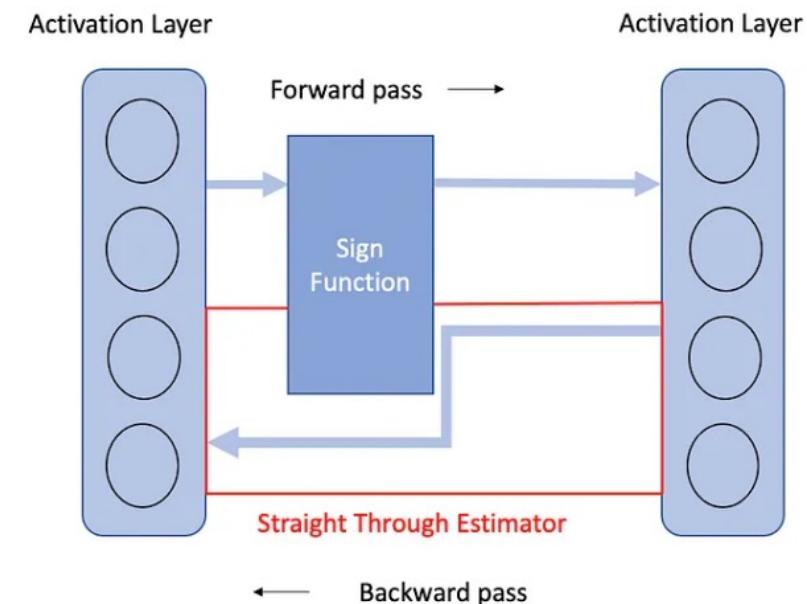
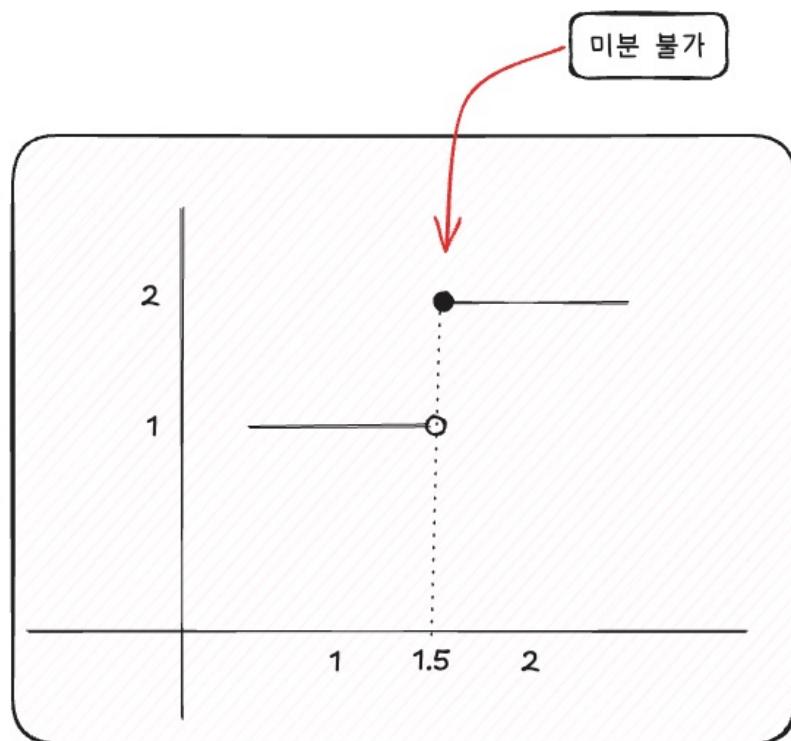


$$[-Q_b, Q_b] (Q_b = 2^{b-1})$$

$$Q_n = -128, Q_p = 127$$

STE(Straight Through Estimator)

역전파 때 미분 불가능한 연산이 포함된 함수에서도 역전파 가능하도록 하는 방법



STE(Straight Through Estimator)

역전파 때 미분 불가능한 연산이 포함된 함수에서도 역전파 가능하도록 하는 방법

```
class BitLinear(nn.Linear):

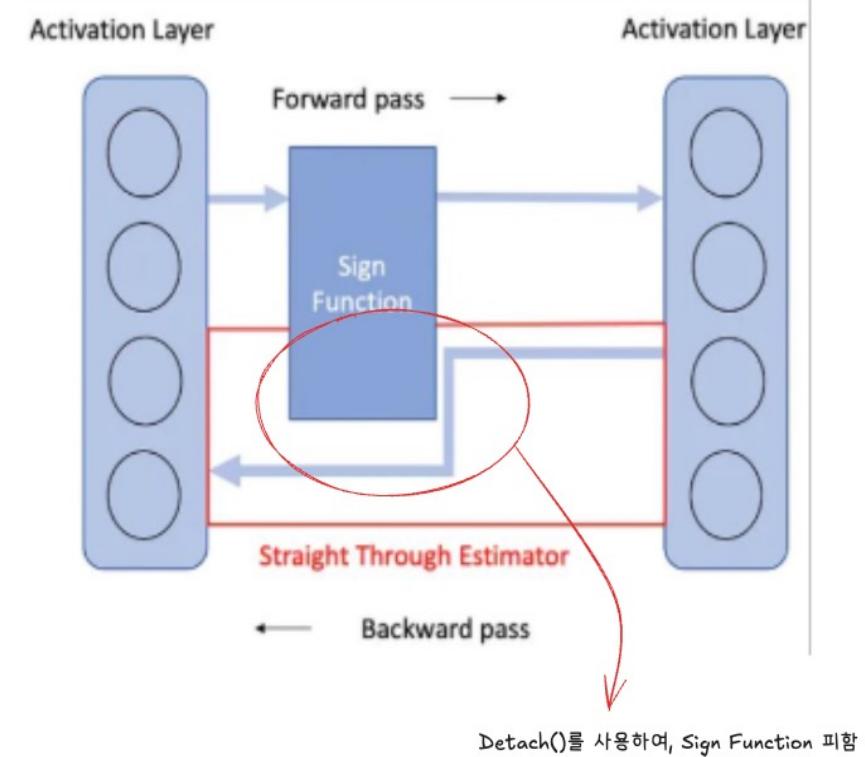
    def __init__(self,
                 *kargs,
                 weight_bits=1,
                 input_bits=8,
                 **kwargs):
        super(BitLinear, self).__init__(*kargs, **kwargs)
        """
        RMSNorm is placed outside BitLinear
        """
        self.weight_bits = weight_bits
        self.input_bits = input_bits

    def forward(self, input):
        quant_input = input + (activation_quant(input, self.input_bits) - input).detach()
        quant_weight = self.weight + (weight_quant(self.weight, self.weight_bits) - self.weight).detach()

        out = nn.functional.linear(quant_input, quant_weight)
        if not self.bias is None:
            out += self.bias.view(1, -1).expand_as(out)

        return out
```

detach() :
텐서를 계산 그래프와 분리하여 그라디언트 전파×



BitNet 추론

Ternary(-1, 0, 1) 양자화 된 가중치와 8bit 활성화를 사용하여, 빠른 행렬곱 연산 통해 전력 소비 절감



BitNet 추론

Ternary(-1, 0, 1) 양자화 된 가중치와 8bit 활성화를 사용하여, 빠른 행렬곱 연산 통해 전력 소비 절감

추론 시

1. 모델 가중치 오프라인에서 1.58bit로 양자화
 2. 입력값 양자화, 스케일
 3. F.linear 연산은 고밀도 연산을 요구하여, 저비트 연산 커널로 대체
 4. 가중치 양자화, 활성화 양자화 스케일링 계수 커널 연산 후 적용
- STE를 따로 구현할 필요 없음

```
import torch
import torch.nn as nn
import torch.nn.functional as F

def activation_quant_inference(x):
    x = LN(x)
    scale = 127.0 / x.abs().max(dim=-1, keepdim=True).values.clamp_(min=1e-5)
    y = (x * scale).round().clamp_(-128, 127)
    return y, scale

class BitLinear(nn.Linear):

    def forward(self, x):
        w = self.weight # weights here are already quantized to (-1, 0, 1)
        w_scale = self.w_scale
        x_quant, x_scale = activation_quant_inference(x)
        y = efficient_kernel(x_quant, w) / w_scale / x_scale
        return y
```

BitNet 추론

Weight Quant, Weight Scale

추론 시

1. 모델 가중치 오프라인에서 1.58bit로 양자화

$$\text{Weight Quant} = \begin{bmatrix} 1 & -1 & 1 \\ -1 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix}$$

$$\text{scale} = \frac{1}{0.8333} \approx 1.2$$

```
import torch
import torch.nn as nn
import torch.nn.functional as F

def activation_quant_inference(x):
    x = LN(x)
    scale = 127.0 / x.abs().max(dim=-1, keepdim=True).values.clamp_(min=1e-5)
    y = (x * scale).round().clamp_(-128, 127)
    return y, scale

class BitLinear(nn.Linear):

    def forward(self, x):
        w = self.weight # weights here are already quantized to (-1, 0, 1)
        w_scale = self.w_scale
        x_quant, x_scale = activation_quant_inference(x)
        y = efficient_kernel(x_quant, w) / w_scale / x_scale
        return y
```

BitNet 추론

Ternary(-1, 0, 1) 양자화 된 가중치와 8bit 활성화를 사용하여, 빠른 행렬곱 연산 통해 전력 소비 절감

추론 시

2. 입력값 양자화, 스케일

$$Absmax = \begin{bmatrix} 127 & -76 & 89 \\ -95 & 42 & -127 \\ 127 & -79 & 48 \end{bmatrix} \quad scale = \begin{bmatrix} 127 \\ 105.83 \\ 158.75 \end{bmatrix}$$

```
import torch
import torch.nn as nn
import torch.nn.functional as F

def activation_quant_inference(x):
    x = LN(x)
    scale = 127.0 / x.abs().max(dim=-1, keepdim=True).values.clamp_(min=1e-5)
    y = (x * scale).round().clamp_(-128, 127)
    return y, scale

class BitLinear(nn.Linear):

    def forward(self, x):
        w = self.weight # weights here are already quantized to (-1, 0, 1)
        w_scale = self.w_scale
        x_quant, x_scale = activation_quant_inference(x)
        y = efficient_kernel(x_quant, w) / w_scale / x_scale
        return y
```

BitNet 추론

커널을 활용해 Weight, Input 연산

추론 시

3. F.linear 연산은 고밀도 연산을 요구하여, 저비트 연산 커널로 대체

| TL1 | | Pack |
|--------|----|------|
| Unpack | | Pack |
| -1 | -1 | 0000 |
| -1 | 0 | 0001 |
| -1 | 1 | 0010 |
| 0 | -1 | 0011 |
| 0 | 0 | 0100 |
| 0 | 1 | 0101 |
| 1 | -1 | 0110 |
| 1 | 0 | 0111 |
| 1 | 1 | 1000 |

| TL2 | | | Pack |
|--------|----|----|--------|
| Unpack | | | Pack |
| -1 | -1 | -1 | 1 1101 |
| -1 | -1 | 0 | 1 1100 |
| -1 | -1 | 1 | 1 1011 |
| -1 | 0 | -1 | 1 1010 |
| ... | | | |
| 0 | 0 | 0 | 0 0000 |
| ... | | | |
| 1 | 0 | 1 | 0 1010 |
| 1 | 1 | -1 | 0 1011 |
| 1 | 1 | 0 | 0 1100 |
| 1 | 1 | 1 | 0 1101 |

| Unpack | Pack |
|--------|------|
| -1 | 00 |
| 0 | 01 |
| 1 | 10 |

I2_S

```
import torch
import torch.nn as nn
import torch.nn.functional as F

def activation_quant_inference(x):
    x = LN(x)
    scale = 127.0 / x.abs().max(dim=-1, keepdim=True).values.clamp_(min=1e-5)
    y = (x * scale).round().clamp_(-128, 127)
    return y, scale

class BitLinear(nn.Linear):

    def forward(self, x):
        w = self.weight # weights here are already quantized to (-1, 0, 1)
        w_scale = self.w_scale
        x_quant, x_scale = activation_quant_inference(x)
        y = efficient_kernel(x_quant, w) / w_scale / x_scale
        return y
```

I2_S: 각각 가중치를 2bit로 압축하여 저장

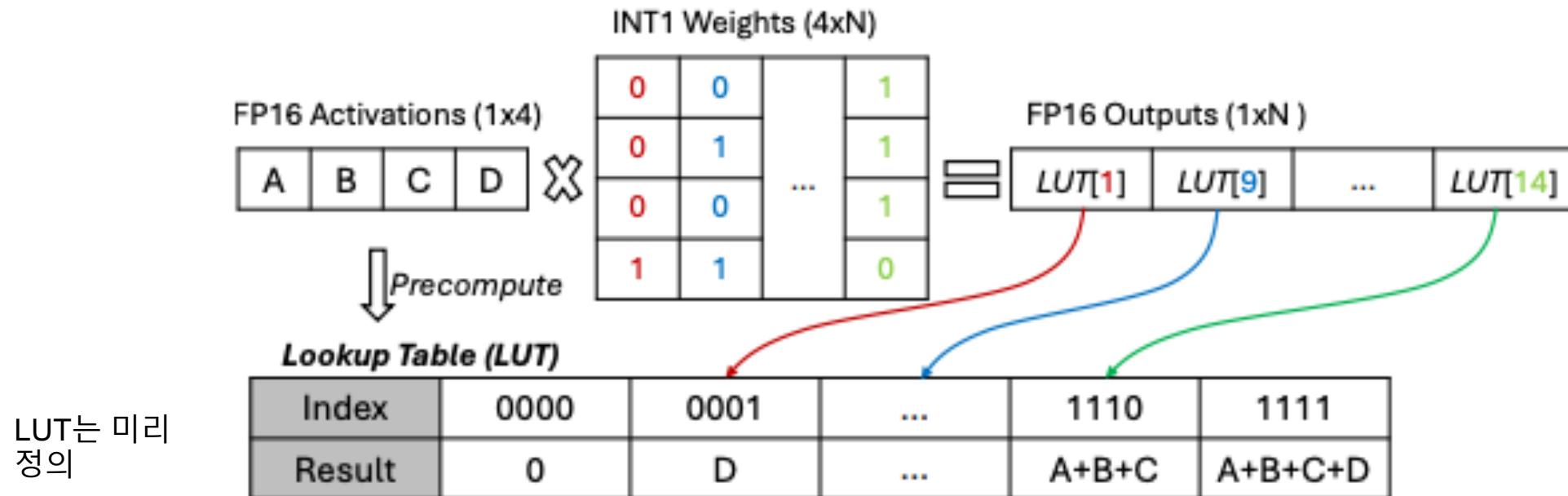
TL1 : 2자리를 4bit 압축

TL2 : 3자리를 5bit 압축

BitNet 추론

커널 연산 어떻게??

1. Lookup Table 계산



추론

환경, 방법

| 추론 환경 | 실험 방법 |
|--|---|
| <ul style="list-style-type: none">• Intel X86_64• CPU(s): 18• RAM : 60GB• Disk : 97GB | <ul style="list-style-type: none">• 단일 질문<ul style="list-style-type: none">◦ 질문50개 배치 단위 속도 비교(TPS, TTFS, 메모리 사용량)◦ 정성적 평가• 키워드 추출<ul style="list-style-type: none">◦ 키워드 추출 정성적 평가 |

추론

단일 질문 추론 속도

Microsoft/bitnet-b1.58-2B-4T, 2.4B, HF_transformers

| | AVG_TTFT | TPS | Memory_Usage |
|--------|----------|----------------|--------------|
| Batch1 | 0.62초 | 7.44 tokens/초 | 4.87GB |
| Batch2 | 1.08초 | 10.76 tokens/초 | 4.91GB |
| Batch3 | 1.2초 | 13.24 tokens/초 | 5.02GB |
| Batch4 | 1.8초 | 15.04 tokens/초 | 5.08GB |

```
max_new_tokens=4096, # 최대  
top_p = 0.9,          # (베이스 모델 default)  
temperature = 0.6,     # (베이스 모델 default)
```

Qwen/Qwen3-0.6B, 0.6B, HF_transformers

| | AVG_TTFT | TPS | Memory_Usage |
|--------|----------|-------------|--------------|
| Batch1 | 0.2초 | 17 tokens/초 | 0.951GB |
| Batch2 | 0.4초 | 24 tokens/초 | 0.954GB |

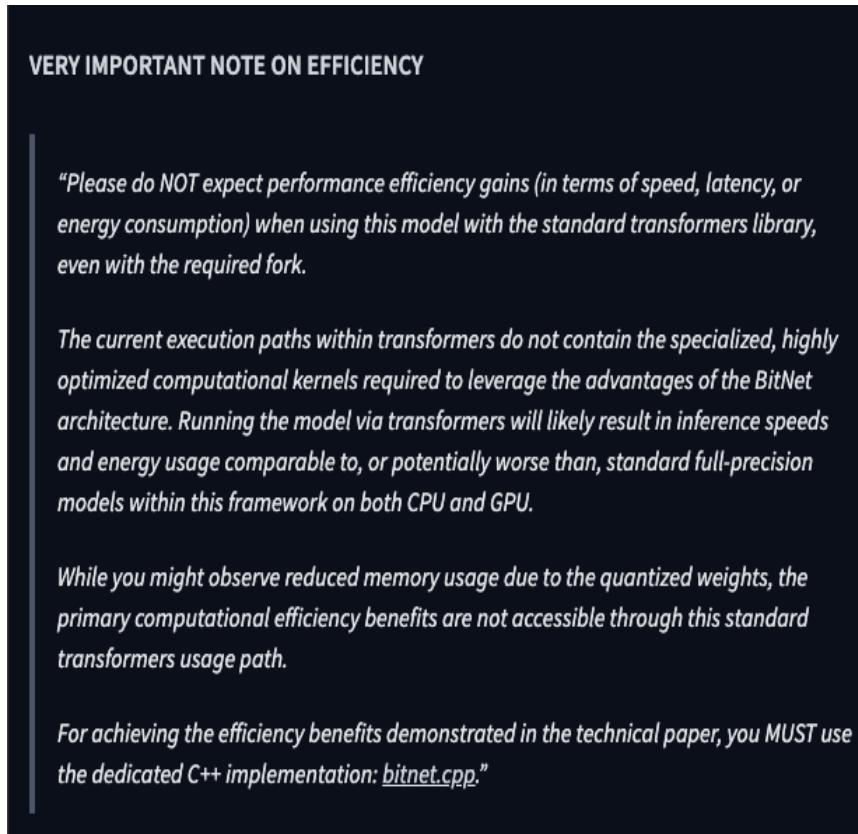
```
max_new_tokens=4096,  
top_p = 0.9,  
temperature = 0.6,
```

질문 예시 50개.

1. 벡터 데이터베이스를 설명해
2. 역전파에 대해 설명 해줘

추론

BitNet with Transformers



HF Transformers 사용 시

- 속도, 지연 시간, 에너지 소비 성능 효율성 기대 x
- 추론 시 최적화 커널을 사용 불가
- Full-precision 연산과 비슷하거나 더 안 좋음
(속도, 지연 시간, 에너지 소비 성능 효율성)

GGUF, Bitnet.cpp

Official Models

| Model | Parameters | CPU | Kernel | | |
|------------------------------------|------------|-----|--------|-----|-----|
| | | | I2_S | TL1 | TL2 |
| BitNet-b1.58-2B-4T | 2.4B | x86 | ✓ | ✗ | ✓ |
| | | ARM | ✓ | ✓ | ✗ |

hf -> gguf

Microsoft / BitNet (Public)

Code Issues 76 Pull requests

main BitNet / utils /

potassiummmm add conversion logic for new r

Name

codegen_t1.py

codegen_t12.py

convert-hf-to-gguf-bitnet.py

convert-ms-to-gguf-bitnet.py

convert.py

e2e_benchmark.py

generate-dummy-bitnet-model.py

kernel_tuning.py

추론BitNet

단일 질문 정성적 평가(영어 질문)

BaseModel Eng

"input": "User: explain vector databases<|eot_id|>Assistant: ",

"output":

A vector database is a specialized type of NoSQL database designed to store and manage high-dimensional vectors efficiently.

These vectors are often used in machine learning, particularly for tasks such as natural language processing (NLP), computer vision, recommendation systems, and other applications that involve similarity search or clustering.

Here is an overview of key aspects of vector databases:

Key Features

1. **High-Dimensional Vectors**: Vector databases handle large datasets where each data point consists of multiple numerical features (dimensions). For example, text embeddings like Word2Vec or GloVe represent words with many dimensions corresponding to their semantic meaning.

2. **Efficient Storage**: They use various techniques to reduce the storage requirements while maintaining performance:

.... 생략

추론 결과

1. 벡터DB에 대해 적절하게 답함

max_new_tokens=4096, # 최대
top_p = 0.9, # (베이스 모델 default)
temperature = 0.6, # (베이스 모델 default)

단일 질문 정성적 평가(영어 질문)

```
"input": "<|im_start|>user\nexplain vector databases<|im_end|>\n<|im_start|>assistant\n",
```

```
"output": "
```

A **Vector Database** is a specialized form of database design used primarily for managing and retrieving data in a structured yet flexible manner. Unlike traditional relational or file-based databases, vector databases store data as **numerical vectors**, where each entry represents a position or feature in a multi-dimensional space.\n\n### Key Concepts:

1. **Vectors**: - Composed of multiple *coordinates* representing specific attributes (e.g., position, temperature, time).

- Example: A 3-D vector might be '(x=10', 'y=20', '-5'), representing spatial orientation.

2. **Data Representation**:

- Documents/entries are transformed into vectors. These vectors contain all necessary characteristics related to the entity being described.\n\n3. **Queries**:
- Queries process vectors to find relationships, similarities, or patterns within collections of vectors. Examples include finding items with high similarity to another, identifying trends over time, or clustering documents based on shared traits.\n\n4. **Indexing & Scalability**:
- No explicit index is required; the system organizes vectors directly for efficient retrieval. This avoids overhead associated with creating static indexes in traditional databases.\n\n5. **Applications**:
- Social media: Sentiment scoring, location tracking, or topic modeling.
- Science research: Analyzing molecular structures or gene sequences.

- Advanced uses involve sparse representations (reducing vector size) for optimization.\n\n### Limitations:
- Performance may vary depending on implementation complexity and hardware.
- Data consistency must ensure all entries align correctly.
In essence, vector dictionaries allow data to be processed more effectively through pattern recognition, enabling applications beyond simple queries. 

max_new_tokens=4096,
top_p = 0.9,
temperature = 0.6,

추론BitNet

단일 질문 정성적 평가(한국어 질문)

BaseModel Kor 질문

"input": "User: 벡터 데이터베이스를 설명해<|eot_id|>Assistant: ",

"output": "벡터数据ベース는 := 데이터베이스와 벡트数据베이스의 성능을 비교한 terminology의 terminology의 한 terminology의 단어 \"vector\"에 대한 terminology의 terme로, 벡터데이터베이스는 := terme의 terminology을 사용한 terminology에서 \"vectors\" 또는 \"vector values\"로 terminology의 térmico가 될 수 있습니다.\n\nVectors, terminology의 term \"vector\", terminology의 Term Definition 2 (TD2)에서 terminology의 TERM ID 4, terminology에서 terminology에서 Term ID 5, terminology에 terminology의 Terminology ID 1, terminology's terminology ID 3, terminology terminology's TERMINOLOGY ID 10, terminology TERM ID, terminology Term ID, Terminology's Terminology 1 (TID1), Terminology Term ID (Term ID), Terminality's Terminally Defined Term (TTD), Terminating Definition 4 (TDD4), Terminator's Terminator (TT), Terminators' Terminators (TTs), Terminations' Terminations

... 생략

문제점

1. 한글 영어 뿐만 아닌 외국어 출력
2. 내용이 적절치 않다

max_new_tokens=4096, # 최대
top_p = 0.9, # (베이스 모델 default)
temperature = 0.6, # (베이스 모델 default)

추론Qwen

단일 질문 정성적 평가(한국어 질문)

"input": "<|im_start|>user\n벡터 데이터베이스를 설명해<|im_end|>\n<|im_start|>assistant\n",

"output":

Vector 데이터베이스의 정의와 기능

Vector 데이터베이스는 다중 차원 공간에서 단일 요소(개체)가 아닌 여러 값을 포함하는 구조(벡터)로 구성된 데이터 저장 시스템입니다. 이들은 다음과 같은 특징을 가집니다:

1. 데이터 유형

각 행은 만약 테이블처럼 한 개의 데이터 포인트라고 생각할 수 있지만,

실제에는 각 행에 다양한 값들이 존재합니다. 예를 들어, 2차원 공간에서는 점으로, 3차원 공간에서는 방향성과 크기를 동시에 표현하는 형태입니다.

2. 차원 및 성질

차수: 하나씩 증가하며, 최대 3차원까지 확장될 수 있습니다.

정렬/구조링크: 단순히 "점" 또는 "문간"의 관계보다는 특정 값의 집합을 유지하고, 이를 통해 분류나 조건 검색 속도를 높이는 역할을 합니다.

비교

항목 전통적인 테이블 Vector 데이터베이스

유저별 처리 중복되는 정보(row) → 별도의 방식 다중값 필터링 가능

쿼리 효율성 고려 필요 없음 다중값 필터링 및 고속 유사도 검색 가능

활용 사례 지오정보학, 머신러닝, 물리적 모델링 추천 시스템, 이미지/텍스트 검색 등

실생활 적용

마사지 치료 시스템: 매번 사용자의 위치와 건강 상태를 측정하기 위해 각 마디의 좌표를 함께 저장해야 하며,

경찰 대응 시스템: 추적되지 않는 경로를 확인하려면 모든 거리를 검토해야 합니다.

Vector 데이터베이스는 백업과 저장 시 효율성을 극대화시키고, 현실 세계에서 더 자연스럽게 데이터를 처리할 수 있도록 도전됩니다! 😊

문제점

1. 할루시네이션 생김

max_new_tokens=4096,
top_p = 0.9,
temperature = 0.6,

추론BitNet

단일 질문 정성적 평가(한국어 질문)

러시아, 중국어, 일본어 토큰 생성 차단

"input": "User: 벡터 데이터베이스를 설명해<|eot_id|>Assistant: ",

"output":

"VECTOR DATASETS BASELINE (Vector Datasets BaseLine)는 ML 모델에 사용되는 벡트 데
이터베이크ett를 terminology으로 설명합니다.

벡트 데이터베이크et= ML에서 중요한 역할을 합니다.

1. **vectors**:

- 벡터 data(db)는 벡터는 ML.Model에 사용됩니다. 벤트는 벨트의 정보를 담는 벤트입니다.
- 예를 들어, 주의한 점에서 벡션은 NMF(b)로 istinguished_data(db)를 생성하는 데 사용됩니다.

2. **Vectors**:

- vectors(data/db): 벡터(data)에 대한 벡팅을 생성합니다.
- These vectors are used to represent the content of text documents in a numerical format that can be processed by machine learning models.

3. **BaseLine**:

- baseLine(): 벡터ml(data/db)가 제공된 정보를 기반으로 ML.Model을 학습하고, 학습된 모델에 적용되기 시작합니다.\n\t- This term emphasizes the importance of using vector datasets as input for training and evaluating machine learning algorithms.

4. **Data**:\n\tIn this context, \"data\" refers to the raw information or texts that need to be transformed into vectors for use with machine learning tasks.\n\t\n\t5. **Training and Evaluation**:\n\tThe Vector Datasets Baseline is often employed during the preprocessing stage of Natural Language Processing (NLP) tasks, such as document classification, sentiment analysis, etc.

...생략

문제점

1. 한글과 영어가 혼용 됨
2. 뒤로 갈 수록 영어로 된 답변만 함

```
max_new_tokens=4096, # 최대
top_p = 0.9,          # (베이스 모델 default)
temperature = 0.6,    # (베이스 모델 default)
repetition_penalty=1.2, # 같은 토큰 반복 방지
no_repeat_ngram_size=3, # 3그램 반복 방지
logits_processor= LogitsProcessorList([BlockerProcessor(tokenizer)])
```

추론BitNet

단일 질문 정성적 평가(한국어 질문)

```
"input": "<|im_start|>user\n벡터 데이터베이스를 설명해<|im_end|>\n<|im_start|>assistant\n",
```

"output": "

벡터 데이터 베어시 (Vector Database)

벡터 데이터 베어시 (Vector Database)는 데이터의 고차원 구조를 저장하고 분석하는 데 사용되는 **비서식적 접근 방식입니다**. 이 기술은 일반적인 관리형 데이터베이는 테이블이나 문서 형태로 이루어진다는 점과 달리 넘어 고차원 공간에서 데이터를 표현하기 위한 유사한 방법을 활용합니다.

주요 특징:

고다변수 집합:

데이터가 여러 차원(예: 좌표, 인코딩된 시각 정보 등)으로 구성되어 있습니다. 예를 들어, 포인트 위치나 시간-series 데이터 등을 단일 벡터로 표현할 수 있습니다.

구조 복잡성 최소화:

전통적으로 테이블 또는 문서 형식으로 데이터를 처리했지만, 벡터 데이터베이지에서는 동적 업데이트 및 **다층 구조 설계** 가능합니다. 변경된 데이터셋에 대한 간단한 수정 없이 유지 maintained 됩니다.

효율적인 검색 및 조건 적용:

특정 패턴(patterns) 찾기와 비교하여 최대화 알고리즘(예: k-Nearest Neighbor) 등의 효율적인 연산 속도를 제공하며, 다양한 계열별 쿼리문에도 적응됩니다.

확장성:

- 하드웨어 크기가 큰 장치에서도 대량의 데이터를 동시에 처리할 수 있으며, 메모리 낭비 문제도 해결되며 성능 향상이 가능합니다.

실무 사례:

추천 시스템: 클릭/매 눈마다 새로운 패턴 생성하여 추천 결과를 개선함.

악마 감정 분류: 고차경우의 데이터 만들기 위해 벡퍼 학습 모델을 통해 정규화 수행.

시간 series: 매일 일자별 행동을 벡거로 표현해 변동성을 반영하도록 설정.

결론적으로, 벡터 데이터베이스는 통용형 데이터 관리 도구보다 더 강력하게 디지털 트렌드를 맞춘 체계이며, 특히 고차분야 데이터 처리 요구사항에 부담 없습니다.

문제점

- 부적절한 표현들이 많다
- 다층 설계 등 추상적 표현 다수

```
max_new_tokens=4096,  
top_p = 0.9,  
temperature = 0.6,  
repetition_penalty=1.2,  
no_repeat_ngram_size=3,  
logits_processor= LogitsProcessorList([BlockerProcessor(tokenizer)])
```

추론BitNet

단일 질문 정성적 평가(한국어 질문)

한국어, 영어 Logit 비율 조정

"input": "User: 벡터 데이터베이스를 설명해<|eot_id|>Assistant: ",

Kor: 0.75
Eng: 0.25

"output":
"VECSEL 데이터베스트는 := 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,
25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40 ... 생략

문제점

1. 한국어 답변 퀄리티 떨어짐

Kor: 0.8
Eng: 0.2

"output":
"VECTOR DATABASE는 벡트 데이터베스트를 terminology으로 부른다.
이는 벡 트에 대한 정보를 저장하고, 데이터의 형식이나 크기를 줄이기 위해 벡팅을 사용하는 방법을 의미합니다.
벡TING은 벡트의 형태로 데이터를 저장하며, 데이터가 변동할 수 있는 경우에도 이를 통해 변형할 수 있습니다.
벡TING의 형성자와 사용법은 다양한 분야에서 사용됩니다.\n\n아래는 벡트 데이터 베이스의 Key Concepts Explained:
1. **구체 데이터와 변형 데이터**:
- **구 Chic**: 데이터의 특정 형식으로 저장되어 있는 데이터입니다.

...생략

Kor: 0.85
Eng: 0.15

"output":
"VECSEL 데이터베ector는 := 데이터베ctor를 사용하여 데이터를 처리하고 분석하는 데 도움을 주는 ettel. 데이터베ectors는 데이터의 특성을 추출하고 이를 통해 데이터를 분석하고, 예측하는 데 사용됩니다.
데이터를 추출한 데이터는 <- 데이터베cket에 저장된 것과 같은 형식으로 저장됩니다.
대신 데이터를 <- 데이터 베ector로 저장할 수 있는 방법:
1. 데이터가 이미 <- 데이터버켓에 저장되어 있기 때문에, 데이터를 데이터베ECTOR에 넣는 것이 불가능합니다.
2. 데이터는 데이터베ktor에 저장되지 않으면서, 데이터들을 데이터베ektor으로 저장하고, 데이터베lector에 넣습니다.

이렇게 하면 데이터베cctor에서 데이터를 읽고, 데이터에 대한 정보를 제공하는 데 usefulness합니다.
예를 들어, 데이터가 데이터베ktion을 통해 추출되면 데이터베CKET에 저장되기 때문에 데이터베tractor에서는 데이터를 불러오고, 분석을 진행할 수 있습니다.

```
max_new_tokens=4096, # 최대  
top_p = 0.9, # (베이스 모델 default)  
temperature = 0.6, # (베이스 모델 default)  
repetition_penalty=1.2, # 같은 토큰 반복 생성 방지  
no_repeat_ngram_size=3, # 3그램 반복 방지  
logits_processor= LogitsProcessorList([BlockerProcessor(tokenizer, korean_ratio=0.85)])
```

```
self.korean_weight = 1.0 + (korean_ratio * 0.5)  
self.english_weight = 1.0 + ((1 - korean_ratio) * 0.5)
```

출처

출처

Wang, Jinheng, et al. "1-bit AI Infra: Part 1.1, Fast and Lossless BitNet b1. 58 Inference on CPUs." *arXiv preprint arXiv:2410.16144* (2024).

Wang, Jinheng, et al. "Bitnet. cpp: Efficient Edge Inference for Ternary LLMs." *arXiv preprint arXiv:2502.11880* (2025).

Ma, Shuming, et al. "BitNet b1. 58 2B4T Technical Report." *arXiv preprint arXiv:2504.12285* (2025).

Ma, Shuming, et al. "The era of 1-bit llms: All large language models are in 1.58 bits." *arXiv preprint arXiv:2402.17764* 1 (2024).

Wang, Hongyu, et al. "Bitnet: Scaling 1-bit transformers for large language models." *arXiv preprint arXiv:2310.11453* (2023).

<https://huggingface.co/1bitLLM>

https://github.com/microsoft/unilm/blob/master/bitnet/The-Era-of-1-bit-LLMs__Training_Tips_Code_FAQ.pdf

<https://huggingface.co/microsoft/bitnet-b1.58-2B-4T>