ДОДАТОК А

Програмний код реалізації методу

У цьому додатку наведено ключові фрагменти програмної реалізації запропонованого методу побудови мовних мікромоделей. Повний вихідний код організований у вигляді Python-пакета micromodel і налічує близько 2500 рядків коду.

А.1 Реалізація функції втрат дистиляції

Лістинг А.1 – Комбінована функція втрат дистиляції знань

```
import torch
import torch.nn.functional as F
class DistillationLoss:
    # Combined distillation loss with QAT.
    # L_total = alpha*L_KL + beta*L_CE + gamma*L_feat +
delta*L_attn
    def __init__(self, alpha=0.6, beta=0.2, gamma=0.15,
delta=0.05,
            temperature=4.0):
        self.alpha = alpha
        self.beta = beta
        self.gamma = gamma
        self.delta = delta
        self.T = temperature
    def kl_loss(self, student_logits, teacher_logits):
        # KL divergence with softmax temperature.
        T = self.T
        p_s = F.log_softmax(student_logits / T, dim=-1)
        p_t = F.softmax(teacher_logits / T, dim=-1)
        return F.kl_div(p_s, p_t, reduction='batchmean') * (T **
```

```
2)
    def ce_loss(self, student_logits, labels):
        return F.cross_entropy(
            student_logits.view(-1, student_logits.size(-1)),

            labels.view(-1),
            ignore_index=-100
        )
    def feat_loss(self, student_hidden, teacher_hidden,
projections):
        # MSE between hidden states after projection.
        loss = 0.0
        for h_s, h_t, proj in zip(student_hidden, teacher_hidden,
projections):
            h_t_proj = proj(h_t)
            loss += F.mse_loss(h_s, h_t_proj)
        return loss / len(student_hidden)
    def attn_loss(self, student_attn, teacher_attn, mapping):
        # Align attention matrices between corresponding layers.
        loss = 0.0
        for l_s, l_t in enumerate(mapping):
            A_s = student_attn[l_s]
            A_t = teacher_attn[l_t]
            if A_s.size(1) != A_t.size(1):
                A_t = A_t.mean(dim=1, keepdim=True).expand_as(A_s)
            loss += F.mse_loss(A_s, A_t)
        return loss / len(mapping)
    def __call__(self, student_out, teacher_out, labels,
                 projections, layer_mapping):
```

```
        L_KL = self.kl_loss(student_out.logits,
teacher_out.logits)
        L_CE = self.ce_loss(student_out.logits, labels)
        L_feat = self.feat_loss(
            student_out.hidden_states, teacher_out.hidden_states,
projections
        )
        L_attn = self.attn_loss(
            student_out.attentions, teacher_out.attentions,
layer_mapping
        )
        L_total = (self.alpha * L_KL + self.beta * L_CE
                + self.gamma * L_feat + self.delta * L_attn)
        return L_total, {
            'L_KL': L_KL.item(), 'L_CE': L_CE.item(),

    'L_feat': L_feat.item(), 'L_attn': L_attn.item()
}
```

А.2 Оцінка важливості шарів для прунінгу

Лістинг А.2 – Обчислення важливості шарів за когерентністю представлень

```
import torch
import torch.nn.functional as F
from tqdm import tqdm
@torch.no_grad()
def compute_layer_importance(model, dataloader, device='cuda'):
    # Estimate layer importance by average non-coherence of hidden
states.
    # I(l) = 1 - mean(cos(h^(l), h^(l-1)))
```

```
    model.eval()
    L = model.config.num_hidden_layers
    cum_cos = torch.zeros(L)
    n_batches = 0
    for batch in tqdm(dataloader, desc='Importance scoring'):
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch.get('attention_mask', None)
        outputs = model(
            input_ids=input_ids,
            attention_mask=attention_mask,
            output_hidden_states=True
        )
        hidden_states = outputs.hidden_states # tuple, len = L+1
        for l in range(L):
            h_prev = hidden_states[l].flatten(0, 1)
            h_curr = hidden_states[l + 1].flatten(0, 1)
            cos = F.cosine_similarity(h_curr, h_prev,
dim=-1).mean()
            cum_cos[l] += cos.cpu()
        n_batches += 1
    mean_cos = cum_cos / n_batches
    importance = 1.0 - mean_cos

    return importance
def select_layers_to_keep(importance, target_num_layers):
    L = importance.size(0)
    if target_num_layers >= L:
        return list(range(L))
    top_indices =
```

```
importance.topk(target_num_layers).indices.sort().values
    return top_indices.tolist()
```

А.3 Головний цикл навчання

Лістинг А.3 – Головний цикл дистиляції з gradual quantization

```
def train_epoch(student, teacher, dataloader, optimizer,
scheduler,
            loss_fn, projections, layer_mapping, epoch,
total_epochs,
                tau_warmup_frac=0.3, device='cuda'):
    # One distillation epoch with gradual ternary quantization.
    student.train()
    teacher.eval()
    total_steps = len(dataloader) * total_epochs
    tau_warmup_steps = int(total_steps * tau_warmup_frac)
    running_loss = 0.0
    for step, batch in enumerate(dataloader):
        global_step = epoch * len(dataloader) + step
        # Update tau (gradual quantization)
        tau = min(1.0, global_step / max(1, tau_warmup_steps))
        for module in student.modules():
            if isinstance(module, BitLinear):
                module.set_quantization_level(tau)
        input_ids = batch['input_ids'].to(device)
        labels = batch['labels'].to(device)
        with torch.no_grad():
            teacher_out = teacher(
                input_ids, output_hidden_states=True,
```

```
            output_attentions=True
        )
    student_out = student(
        input_ids, output_hidden_states=True,
        output_attentions=True
    )
    loss, components = loss_fn(
        student_out, teacher_out, labels,
        projections, layer_mapping
    )
    optimizer.zero_grad()
    loss.backward()
    torch.nn.utils.clip_grad_norm_(student.parameters(),
max_norm=1.0)
    optimizer.step()
    scheduler.step()
    running_loss += loss.item()
    if step % 100 == 0:
        print(f'[Epoch {epoch}, step {step}]
loss={loss.item():.4f} '
              f'tau={tau:.3f} L_KL={components["L_KL"]:.3f}')
    return running_loss / len(dataloader)
```

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	84 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Артеменко О.І.			Метод побудови мовних мікро-моделей на основі LLM та методу їх екстремального стиснення	Шифр групи	Код напр./спец.
Перевірив		Павленко Є.П.				СШІздм-24-1	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Чала Л.Е.					