ДОДАТОК А

Матеріали апробації, публікацій та програмної реалізації

У додатку наведено супровідні матеріали, що підтверджують апробацію результатів кваліфікаційної роботи та підготовку публікацій за темою дослідження.

До матеріалів апробації належать тези доповіді «Порівняльний аналіз сучасних методів ефективного за параметрами донавчання LLM», підготовлені для 30-го Міжнародного молодіжного форуму «Радіоелектроніка та молодь у XXI столітті».

До матеріалів публікації належить англомовна стаття «Spectral Gated Adapter: Parameter-Efficient Fine-Tuning with Gated Spectral Filters», підготовлена відповідно до вимог REKS.

Супровідні файли: SPECTRAL GATED ADAPTER article.pdf, матеріали ММФ, trebov_REKS.

Фрагмент програмної реалізації Spectral Gated Adapter наведено в лістингу А.1. Код подано в додатку, щоб зберегти цілісність основного викладу та водночас показати ключові елементи відтворення методу.

Лістинг А.1 – Фрагмент програмної реалізації Spectral Gated Adapter

```
class SpectralGatedAdapter(nn.Module):
    """
    Novel adapter with spectral regularisation
    and token-wise gating.
    """

    def __init__(
        self,
        hidden_size: int,
        rank: int = 16,
        dropout: float = 0.1,
        use_gate: bool = True,
        alpha_init: float = 1.0,
        temperature_init: float = 0.0,
        use_residual_ln: bool = False,
    ):
        super().__init__()
        self.rank = rank
        self.use_gate = use_gate
```

```
        self.use_residual_ln = use_residual_ln
        # Cache identity for spectral penalty
        self.register_buffer(
            "_identity", torch.eye(rank), persistent=False
        )
        self.down = nn.Linear(hidden_size, rank, bias=False)
        self.act = nn.SiLU()
        self.up = nn.Linear(rank, hidden_size, bias=False)
        if self.use_gate:
            self.pre_gate = nn.LayerNorm(hidden_size)
            self.gate = nn.Linear(
                hidden_size, rank, bias=True
            )
        else:
            self.pre_gate = None
            self.gate = None
        self.dropout = nn.Dropout(dropout)
        self.residual_scale = nn.Parameter(
            torch.tensor(alpha_init, dtype=torch.float32)
        )
        self.temperature = nn.Parameter(
            torch.tensor(
                temperature_init, dtype=torch.float32
            )
        )
        self.residual_ln = (
            nn.LayerNorm(hidden_size)
            if self.use_residual_ln
            else None
        )
        self.last_gate_values = None
        self.reset_parameters()

    def reset_parameters(self) -> None:
        nn.init.kaiming_uniform_(
            self.down.weight, a=math.sqrt(5)
        )
        nn.init.zeros_(self.up.weight)
        if self.use_gate and self.gate is not None:
            nn.init.zeros_(self.gate.weight)
            nn.init.zeros_(self.gate.bias)

    def forward(
        self, hidden_states: torch.Tensor
    ) -> torch.Tensor:
        source_states = (
            self.residual_ln(hidden_states)
            if self.use_residual_ln
            and self.residual_ln is not None
            else hidden_states
        )
        down_proj = self.down(source_states)
        if (
```

```
            self.use_gate
            and self.gate is not None
            and self.pre_gate is not None
        ):
            gate_inputs = self.pre_gate(source_states)
            # Shape: [Batch, Seq, Rank]
            gate_vals = torch.sigmoid(
                self.gate(gate_inputs)
                * torch.exp(self.temperature)
            )
            if not self.training:
                self.last_gate_values = (
                    gate_vals.detach()
                )
        else:
            gate_vals = 1.0
        modulated = self.dropout(
            self.act(down_proj) * gate_vals
        )
        up_proj = self.up(modulated)
        return self.residual_scale * self.dropout(
            up_proj
        )

    def spectral_penalty(self) -> torch.Tensor:
        identity = self._identity
        if (
            identity.device != self.down.weight.device
            or identity.dtype != self.down.weight.dtype
        ):
            identity = identity.to(
                device=self.down.weight.device,
                dtype=self.down.weight.dtype,
            )
        down_cov = (
            self.down.weight @ self.down.weight.T
        )
        up_cov = self.up.weight.T @ self.up.weight
        penalty_down = torch.norm(
            down_cov - identity, p="fro"
        )
        penalty_up = torch.norm(
            up_cov - identity, p="fro"
        )
        penalty = penalty_down + penalty_up
        if not torch.isfinite(penalty):
            raise FloatingPointError(
                "Non-finite spectral penalty."
            )
        return penalty
```

Лістинг А.2 – Фрагмент програмної реалізації

```
def inject_sga_adapters(
    model: AutoModelForSequenceClassification,
```

```
    rank: int,
    dropout: float,
    use_gate: bool = True,
    alpha_init: float = 1.0,
    temperature_init: float = 0.0,
    use_residual_ln: bool = False,
) -> List[SpectralGatedAdapter]:
    adapters = []
    for layer in model.bert.encoder.layer:
        adapter = SpectralGatedAdapter(
            model.config.hidden_size,
            rank=rank,
            dropout=dropout,
            use_gate=use_gate,
            alpha_init=alpha_init,
            temperature_init=temperature_init,
            use_residual_ln=use_residual_ln,
        )
        layer.output.add_module(
            "sga_adapter", adapter
        )
        original_fwd = layer.output.forward

        def forward_with_adapter(
            self, hidden_states, input_tensor,
            original_fwd=original_fwd,
            adapter=adapter
        ):
            hidden_states = original_fwd(
                hidden_states, input_tensor
            )
            return hidden_states + adapter(
                hidden_states
            )

        layer.output.forward = types.MethodType(
            forward_with_adapter, layer.output
        )
        adapters.append(adapter)

    # Freeze backbone, unfreeze adapters + head
    for param in model.parameters():
        param.requires_grad = False
    for adapter in adapters:
        for param in adapter.parameters():
            param.requires_grad = True
    for param in model.classifier.parameters():
        param.requires_grad = True

    return adapters
```

Лістинг А.3 – Фрагмент програмної реалізації

```
class SpectralAdapterTrainer(Trainer):
```

```
def __init__(
    self,
    *args,
    spectral_lambda: float = 1e-4,
    adapter_modules: List[
        SpectralGatedAdapter
    ] = None,
    **kwargs,
):
    super().__init__(*args, **kwargs)
    self.spectral_lambda = spectral_lambda
    self.adapter_modules = (
        adapter_modules or []
    )

def compute_loss(
    self, model, inputs, return_outputs=False
):
    outputs = model(**inputs)
    loss = outputs.loss

    if (
        self.spectral_lambda > 0
        and self.adapter_modules
    ):
        penalty = torch.zeros_like(loss)
        for adapter in self.adapter_modules:
            penalty = (
                penalty
                + adapter.spectral_penalty()
            )
        loss = (
            loss
            + self.spectral_lambda * penalty
        )

    return (
        (loss, outputs)
        if return_outputs
        else loss
    )
```

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	68 с.
	Інші документи	
2.	Презентаційні матеріали	_16_ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Мічурін І.Є.			Методи ефективного донавчання великих мовних моделей із вдосконаленням механізмів адаптерів	Шифр групи	Код напр./спец.
Перевірив		Рябова Н. В.				СШІм-24-1	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Чала Л.Е.					