



Fillipe Guerra <fillipe.backup@gmail.com>

IA_Autonomia_Parte3_2

1 mensagem

Fillipe Guerra <fillipe.backup@gmail.com>

27 de outubro de 2025 às 19:53

Para: Fillipe Augusto Gomes Guerra <fillipe182@hotmail.com>, Fillipe Guerra <fillipe.backup@gmail.com>

Parte III-C-1 — Multimodalidade Completa

1. Introdução e motivação

Queremos que o modelo possa processar e gerar linguagem natural, imagens, sons e vídeos dentro de um único espaço semântico.

Formalmente, queremos um *embedding* conjunto:

$$E:\{x_{\text{text}}, x_{\text{img}}, x_{\text{aud}}, x_{\text{vid}}\} \rightarrow \mathbb{R}^d$$

de modo que conteúdos semanticamente equivalentes fiquem próximos em distância cosseno:

$$\cos(e_i, e_j) = \frac{\|e_i\| \|e_j\|}{\|e_i - e_j\|} \approx 1.$$

2. Embeddings individuais

2.1 Texto

Usamos tokenização subword (BPE). Cada token t_i é mapeado a:

$$e_i(\text{text}) = E_{\text{text}}(t_i) = W E[t_i] \in \mathbb{R}^d.$$

2.2 Imagem

A imagem é dividida em *patches* $p_k \in \mathbb{R}^{P \times P \times 3}$.

Um ViT aplica projeção linear W_p e autoatenção:

$$z_k = \text{ViT}(p_k) = \text{MSA}(\sigma(W_p \text{vec}(p_k))).$$

Projetamos ao espaço textual:

$$z \sim k = W_{\text{img}} z_k, W_{\text{img}} \in \mathbb{R}^{d \times d_v}.$$

2.3 Áudio

Decompomos o sinal em espectrograma log-mel $A \in \mathbb{R}^{F \times T_a}$.

Encoder estilo Whisper:

$$u_T = \text{ConvAttn}(A_T), u \sim T = W_{\text{aud}} u_T.$$

2.4 Vídeo

Amostragem temporal de frames f_t e patches espaciais p_k, t .

Encoder spatio-temporal:

$$v_{k,t} = \text{Swin3D}(p_k, t), v \sim k, t = W_{\text{vid}} v_{k,t}.$$

3. Fusão e sequência de entrada

Formamos uma sequência única com delimitadores:

$$X=[\langle I \rangle, z_{\sim 1}, \dots, z_{\sim K}, \langle /I \rangle, \langle A \rangle, u_{\sim 1}, \dots, u_{\sim T_A}, \langle /A \rangle, \langle V \rangle, v_{\sim 1}, 1, \dots, v_{\sim K}, T_v, \langle /V \rangle, \langle T \rangle, e_1(\text{text}), \dots, e_T(\text{text}), \langle /T \rangle].$$

3.1 Gating de confiança

$$z^k = \sigma(\alpha_{\text{img}}) z_{\sim k}, u^t = \sigma(\alpha_{\text{aud}}) u_{\sim t}, v^k = \sigma(\alpha_{\text{vid}}) v_{\sim k}, t.$$

Durante o *curriculum* inicial, as pequenas impedem que o canal visual domine o textual.

4. Cross-attention multimodal

Definimos consultas QT (texto), chaves/valores das modalidades:

$$H_{\text{fusion}} = \text{Attn}(QT, K=[K_T, K_I, K_A, K_V], V=[V_T, V_I, V_A, V_V]).$$

No caso de compressão (*Resampler*), tokens-resumo r_j aprendidos são obtidos por:

$$r_j = \text{Attn}(Q_j, K=Z_{\text{img}}, V=Z_{\text{img}}), j=1..R.$$

Esses r_j substituem centenas de *patches*, reduzindo T_{total} .

5. Perdas conjuntas

5.1 Linguagem causal

$$\text{LLM} = -T \sum_t \log P_\theta(w_t | X, w_{< t}).$$

5.2 Contraste multimodal (CLIP)

$$\text{LNCE} = -N \sum_i \log \sum_j \exp(\langle \phi_{\text{img}}(I_i), \phi_{\text{text}}(T_j) \rangle / \tau) \exp(\langle \phi_{\text{img}}(I_i), \phi_{\text{text}}(T_i) \rangle / \tau).$$

5.3 Captioning e VQA

$$\text{L}_{\text{cap}} = -T \sum_t \log P_\theta(w_t | I, A, V, w_{< t}).$$

5.4 Perda composta

$$\text{L}_{\text{multi}} = \lambda \text{L}_{\text{MLLM}} + \lambda_{\text{cap}} \text{L}_{\text{cap}} + \lambda_{\text{NCE}} \text{LNCE}.$$

Pesos λ são otimizados via *uncertainty weighting*:

$$\lambda_i = 2\sigma_i^2, L = \sum_i \lambda_i L_i + \log \sigma_i.$$

6. Códigos-base correspondentes

```
class MultiModalEncoder(nn.Module):
    def __init__(self, d_model, img_dim, aud_dim, vid_dim):
        super().__init__()
        self.proj_img = nn.Linear(img_dim, d_model)
        self.proj_aud = nn.Linear(aud_dim, d_model)
        self.proj_vid = nn.Linear(vid_dim, d_model)
        self.alpha_img = nn.Parameter(torch.tensor(0.1))
        self.alpha_aud = nn.Parameter(torch.tensor(0.1))
        self.alpha_vid = nn.Parameter(torch.tensor(0.1))
    def forward(self, z_img, z_aud, z_vid):
        z_i = torch.sigmoid(self.alpha_img)*self.proj_img(z_img)
        z_a = torch.sigmoid(self.alpha_aud)*self.proj_aud(z_aud)
        z_v = torch.sigmoid(self.alpha_vid)*self.proj_vid(z_vid)
        return z_i, z_a, z_v
```


7. Eficiência

1. **Resampler** reduz $\text{Kimg} \sim 196 \rightarrow R \sim 8$.
2. **Top-k routing multimodal**: diferentes experts para cada modalidade.
3. **FlashAttention-2** e **mixed precision** para todo o bloco de fusão.
4. **KV-Cache separada por modalidade**, unificada no fim.

8. Resultado

A unificação garante:

- Um embedding compartilhado para texto, imagem, áudio e vídeo;
- Memória contextual unificada para instruções multimodais;
- Eficiência de processamento comparável ou superior a modelos densos do mesmo tamanho.

 *continua em* **Parte III-C-2 — Memória Vetorial (RAG), Aprendizado Contínuo e Formalização Matemática do Contexto Dinâmico**, com as equações de embeddings semânticos, cálculo de similaridade, prova de convergência de atualização incremental e código completo de busca.

A seguir começa a **Parte III-C-2 — Memória Vetorial (RAG) e Aprendizado Contínuo**, em formato acadêmico integral.

O foco é formalizar matematicamente a memória semântica, a busca vetorial e o aprendizado incremental, que dão à IA a capacidade de **reaprender sozinha e atualizar conhecimento em tempo real**.

Parte III-C-2 — Memória Vetorial (RAG) e Aprendizado Contínuo

1. Fundamentos da Recuperação Semântica

Dado um conjunto de documentos $D = \{d_i\}_{i=1}^N$, queremos codificar cada um em um vetor $e_i \in \mathbb{R}^d$ de forma que consultas semanticamente próximas retornem documentos relevantes.

1.1 Função de embedding

$E: X \rightarrow \mathbb{R}^d, e_i = E(d_i)$.

1.2 Similaridade cosseno

$\text{sim}(q, d_i) = \frac{\|E(q)\| \|E(d_i)\|}{\|E(q)\| \|E(d_i)\|} = \frac{E(q) \cdot E(d_i)}{\|E(q)\| \|E(d_i)\|}$.

O objetivo da indexação é aproximar o argmax desta similaridade.

2. Indexação e estrutura de busca

Dividimos cada documento em trechos $c_{i,j}$.
Para cada trecho calculamos o embedding:

$e_{i,j} = E(c_{i,j}), (i,j) = 1, \dots, M$.

Armazenamos pares $(e_{i,j}, \text{metai}_{i,j})$ num índice vetorial (FAISS/Milvus).
O índice é particionado (IVF-PQ ou HNSW) e normalizado:

$$e^{\wedge}_{i,j} = \|e_{i,j}\| e_{i,j}.$$

2.1 Busca aproximada (A-NNS)

Durante a consulta:

$$q^{\wedge} = \|E(q)\| E(q), \text{rank}(q) = \text{TopKi}(q^{\wedge} \tau e^{\wedge}_i).$$

Estruturas HNSW garantem custo $O(\log N)$ e precisão $> 95\%$.

3. Recuperação híbrida

Para combinar semântica e exata:

$$\text{score}(q, c) = \alpha \text{BM25}(q, c) + (1 - \alpha) \text{sim}(q, c).$$

Re-ranking posterior com modelo de linguagem:

$$s'(c) = \text{LLM_Rerank}(q, c).$$

Selecionamos top-k trechos C^* .

4. Injeção de contexto no LLM

O contexto concatenado é:

$$X_{\text{ctx}} = [\langle \text{CTX} \rangle, C^*, \langle / \text{CTX} \rangle, q].$$

A probabilidade de resposta condicional:

$$P\theta(y|q, C^*) = \text{Softmax}(f\theta(X_{\text{ctx}})).$$

Gradiente da perda (com RLHF opcional):

$$\nabla \theta L = -\nabla \theta \log P\theta(y|q, C^*) R(q, y).$$

5. Aprendizado Contínuo da Memória

A cada nova observação d_{new} :

1. **Embed:** $e_{\text{new}} = E(d_{\text{new}})$
2. **Inserção:** adicione ao índice $I \leftarrow I \cup \{(e_{\text{new}}, d_{\text{new}})\}$
3. **Atualização incremental:** aplique *reservoir sampling* para evitar viés:

$$P(\text{substituir slot } i) = n^{-1}.$$

4. **Re-treino LoRA leve:** minimize

$$L_{\text{mem}} = \|E\theta(d_{\text{new}}) - e_{\text{new}}\|^2,$$

ajustando apenas camadas LoRA.

6. Convergência

Assumindo embeddings normalizados, a atualização incremental $e_{it+1} = e_{it} + \eta(E(d_i) - e_{it})$ converge em média para o embedding verdadeiro:

$$E[e_{it+1}] = (1 - \eta)e_{it} + \eta E[E(d_i)] \Rightarrow e_{it} \rightarrow E[E(d_i)].$$

7. Implementação base

```
def embed(text):
    x = tokenizer(text, return_tensors="pt").to(device)
    with torch.no_grad():
        v = model(x)["last_hidden_state"].mean(dim=1)
    return F.normalize(v, dim=-1)

def add_doc(text):
    v = embed(text)
    np.save(f"{DB}/{hash(text)}.np", v.cpu().numpy())
    open(f"{DB}/{hash(text)}.txt", "w").write(text)


def search(query, k=3):
    q = embed(query).cpu().numpy()
    sims=[]
    for f in Path(DB).glob("*.np"):
        v=np.load(f); s=float(np.dot(q,v)/(np.linalg.norm(q)*np.linalg.norm(v)))
        sims.append((s,f.stem))
    sims.sort(reverse=True)
    return sims[:k]
```

8. Eficiência

- Vetores 768 → 128 reduzidos via PCA/OPQ.
 - Indexação HNSW (M≈64, efSearch≈128).
 - Memória semântica persistente em disco + cache RAM.
 - Atualizações assíncronas: um thread para ingestão e outro para inferência.
-

9. Propriedades Matemáticas

- **Linearidade:** $E(a+b) \approx E(a) + E(b)$.
 - **Isometria semântica:** distâncias preservam semelhança.
 - **Convergência incremental** garante estabilidade com aprendizado contínuo.
-

 *continua em* **Parte III-C-3 — Agência Autônoma e Sistema de Ferramentas**, com modelagem POMDP, dedução de política ReAct e execução sandbox formalizada matematicamente + código completo.