Deepseek 文档生成结果样例

智能体系统数学形式化与架构设计文档

1. 核心数学建模（复杂公式与符号说明）

设智能体在离散时间步 t 下与环境交互。定义通用状态空间 $\mathcal{S}$，动作空间 $\mathcal{A}$，观测空间 $\mathcal{O}$。

公式 1：部分可观测马尔可夫决策过程（POMDP）核心转移

$$b_{t+1}(s') = \eta \cdot \underbrace{\sum_{s \in \mathcal{S}} P(s'|s, a_t)}_{\text{状态转移}} \cdot \underbrace{\sum_{o \in \mathcal{O}} O(o|s', a_t) \cdot \mathbb{I}(o = o_t)}_{\text{观测似然}} \cdot b_t(s)$$

说明：$b_t(s)$ 为信念状态，η 为归一化常数，P 为转移概率，O 为观测函数，$\mathbb{I}$ 为指示函数。

公式 2：分层强化学习（HRL）期权策略梯度

$$\nabla J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot \underbrace{\sum_{k=t}^{T} \gamma^{k-t} r_k}_{\text{折扣回报}} + \underbrace{\beta \cdot \nabla_\theta \mathcal{H}(\pi_\theta)}_{\text{熵正则项}} \right]$$

公式 3：多头自注意力机制（Transformer 核心）

$$\text{Attention}(Q,K,V)=\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V,\quad \text{MultiHead}=\text{Concat}(\text{head}_1,\ldots,\text{head}_h)W^O$$

其中 $Q=XW^Q, K=XW^K, V=XW^V$，d_k 为键维度，W 为可学习权重矩阵。

公式 4：世界模型中的潜在动力学预测

$$\mathcal{L}_{\text{World}}=\underbrace{\mathbb{E}_{z_t\sim q_\phi}\left[\log p_\psi(o_t|z_t)\right]}_{\text{重建损失}}-\underbrace{D_{KL}\left(q_\phi(z_t|o_t,z_{t-1},a_{t-1})\parallel p_\theta(z_t|z_{t-1},a_{t-1})\right)}_{\text{先验-后验 KL 散度}}$$

公式 5：多智能体协作的加权双 Q 学习（VDN 混合）

$$Q_{\text{tot}}(\tau,\mathbf{u})=\sum_{i=1}^{N}\alpha_i\cdot Q_i(\tau_i,u_i)+\underbrace{\lambda\cdot\sum_{i<j}\beta_{ij}\cdot Q_iQ_j}_{\text{成对交互项}}$$

公式 6：逆强化学习（IRL）的最大熵目标

$$\mathcal{L}_{\text{IRL}}=\max_{\psi}\mathbb{E}_{\tau\sim\mathcal{D}_{\text{demo}}}\left[\sum_t r_\psi(s_t,a_t)\right]-\log\sum_{\tau'\in\text{Traj}}\exp\left(\sum_t r_\psi(s'_t,a'_t)\right)$$

公式 7：连续控制中的 MPC 代价函数

$$J(\mathbf{u}_{0:H}) = \underbrace{(x_H - x_{\text{ref}})^T Q_f (x_H - x_{\text{ref}})}_{\text{终端代价}} + \sum_{t=0}^{H-1} \left[(x_t - x_{\text{ref}})^T Q (x_t - x_{\text{ref}}) + u_t^T R u_t + \Delta u_t^T \right.$$

公式 8：策略熵约束优化（SAC 风格）

$$\pi^* = \arg\min_{\pi} D_{KL} \left(\pi(\cdot|s_t) \parallel \frac{\exp(Q_{\text{soft}}(s_t, \cdot)/\alpha)}{Z(s_t)} \right), \quad \text{其中} Q_{\text{soft}} = r + \gamma \mathbb{E}[V_{\text{soft}}]$$

公式 9：元学习（MAML）的内外循环更新

$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\text{task}_i}(f_\theta), \quad \min_\theta \sum_{i=1}^{M} \mathcal{L}_{\text{task}_i}(f_{\theta_i'})$$

公式 10：多模态融合（跨注意力门控机制）

$$\mathcal{F}_{\text{fusion}} = \sigma\left(W_g[\text{TextEnc}; \text{ImageEnc}; \text{LidarEnc}]\right) \odot \left(\sum_{m \in \mathcal{M}} \lambda_m \cdot \text{CrossAttn}(Q_m, K_m, V_m) \right)$$

公式 11：安全约束的拉格朗日松弛（CMDP）

$$\min_{\pi} \mathbb{E}\left[\sum \gamma^t r_t\right] \quad \text{s.t.} \quad \mathbb{E}\left[\sum \gamma^t c_t\right] \leq d, \quad \mathcal{L}(\pi, \mu) = J_r(\pi) + \mu(J_c(\pi) - d)$$

所有公式符号总说明表：

符号	说明	符号	说明
s, s', s_t	状态及下一状态	a_t, u_t	动作/控制输入
π_θ	策略网络（参数 θ）	Q, K, V	注意力查询、键、值
γ	折扣因子 (0~1)	α, β, λ	权重系数或温度参数
$\mathcal{H}$	熵函数	D_{KL}	KL 散度
q_ϕ, p_ψ	编码器/解码器分布	τ	轨迹序列
W^Q, W^K	线性投影矩阵	$\odot$	逐元素乘积
σ	Sigmoid 激活函数	μ	拉格朗日乘子

2. 复杂表格

表格 1：多智能体协作场景下的算法性能对比（含统计指标）

算法	环境	平均回报 (Mean±Std)	收敛步数 (万)	通信带宽 (MB/s)	鲁棒性评分	成功率 (Hard 模式)
MAPPO	StarCraft II	94.2 ± 3.1	120	0.8	7.2/10	87.3%
QMIX	Multi-Particle	82.7 ± 5.4	85	0.2	6.5/10	76.1%
VDN	Multi-Particle	75.3 ± 6.2	70	0.15	5.9/10	68.4%
HATPRO	3D 导航	101.5 ± 2.8	200	2.4	9.1/10	94.6%

算法	环境	平均回报 (Mean±Std)	收敛步数 (万)	通信带宽 (MB/s)	鲁棒性评分	成功率 (Hard 模式)
ICQ	混合博弈	88.0 ± 4.0	95	0.6	8.0/10	81.2%

表格 2：世界模型架构组件与参数量（多模态版）

组件名称	输入模态	编码器类型	潜在维度	参数量 (M)	输出用途
视觉编码器	RGB 图像 (224x224)	ResNet-50 + ViT	512	86.2	隐状态 z_t
点云编码器	LiDAR (16 通道)	PointNet++	256	34.7	几何特征
时序记忆模块	动作序列	GRU + 神经图灵机	1024	120.5	时序信用分配
奖励解码器	联合隐状态	3 层 MLP (1024-512-1)	-	5.3	奖励预测 $\hat{r}$
动态预测器	z_t, a_t	Transformer Decoder	512	78.9	下一隐状态 z_{t+1}
对比投影头	隐状态	2 层 MLP + 归一化	128	2.1	对比损失计算

3. 复杂代码（Python + PyTorch 风格，含类型注解）

```
import torch
import torch.nn as nn
```

```
import torch.nn.functional as F
from typing import Tuple, Optional, List
from dataclasses import dataclass

@dataclass
class HyperParameters:
    """ 复杂超参数配置"""
    d_model: int = 512
    n_heads: int = 8
    d_ff: int = 2048
    dropout: float = 0.1
    max_seq_len: int = 1024

class MultiHeadCrossAttention(nn.Module):
    """ 带门控的多模态交叉注意力模块（对应公式 10）"""
    def __init__(self, hp: HyperParameters, num_modalities: int = 3):
        super().__init__()
        self.num_heads = hp.n_heads
        self.d_k = hp.d_model // hp.n_heads
        self.w_q = nn.Linear(hp.d_model, hp.d_model, bias=False)
        self.w_k = nn.Linear(hp.d_model, hp.d_model, bias=False)
        self.w_v = nn.Linear(hp.d_model, hp.d_model, bias=False)
        self.w_o = nn.Linear(hp.d_model, hp.d_model, bias=False)
        self.gate_mlp = nn.Sequential(
            nn.Linear(hp.d_model * num_modalities, hp.d_model),
            nn.ReLU(),
            nn.Linear(hp.d_model, num_modalities),
            nn.Sigmoid()
        )
        self.dropout = nn.Dropout(hp.dropout)
```

```
    def forward(self, query: torch.Tensor, key: torch.Tensor,
                value: torch.Tensor, attn_mask: Optional[torch.Tensor] = None) -> torch.Tensor:
        # 线性变换并拆分为多头
        Q = self.w_q(query).view(query.size(0), -1, self.num_heads, self.d_k).transpose(1, 2)
        K = self.w_k(key).view(key.size(0), -1, self.num_heads, self.d_k).transpose(1, 2)
        V = self.w_v(value).view(value.size(0), -1, self.num_heads, self.d_k).transpose(1, 2)

        # 缩放点积注意力
        scores = torch.matmul(Q, K.transpose(-2, -1)) / (self.d_k ** 0.5)
        if attn_mask is not None:
            scores = scores.masked_fill(attn_mask == 0, -1e9)
        attn_weights = F.softmax(scores, dim=-1)
        attn_weights = self.dropout(attn_weights)

        context = torch.matmul(attn_weights, V)
        context = context.transpose(1, 2).contiguous().view(query.size(0), -1, self.d_k * self.num_heads
        return self.w_o(context)

class SACWithLagrangian(nn.Module):
    """ 带安全约束的 SAC 算法片段（对应公式 8 与 11）"""
    def __init__(self, state_dim: int, action_dim: int, hidden_dim: int = 256):
        super().__init__()
        self.actor = nn.Sequential(
            nn.Linear(state_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, 2 * action_dim)  # 均值和 log 标准差
        )
```

```
    self.critic = nn.Sequential(
        nn.Linear(state_dim + action_dim, hidden_dim),
        nn.ReLU(),
        nn.Linear(hidden_dim, 1)
    )
    self.log_alpha = nn.Parameter(torch.zeros(1, requires_grad=True))
    self.log_mu = nn.Parameter(torch.zeros(1, requires_grad=True)) # 拉格朗日乘子

def get_action(self, state: torch.Tensor) -> Tuple[torch.Tensor, torch.Tensor]:
    mean, log_std = self.actor(state).chunk(2, dim=-1)
    std = torch.exp(log_std.clamp(-20, 2))
    normal = torch.distributions.Normal(mean, std)
    z = normal.rsample()
    action = torch.tanh(z) # 双曲正切限制动作范围
    log_prob = normal.log_prob(z) - torch.log(1 - action.pow(2) + 1e-6)
    return action, log_prob.sum(dim=-1)

def compute_cost_penalty(self, cost: torch.Tensor, threshold: float) -> torch.Tensor:
    """拉格朗日惩罚项计算"""
    lagrangian = torch.exp(self.log_mu)
    return lagrangian * (cost - threshold)
```

4. Mermaid 图表集（≥8 种不同类型）

图 1：系统架构流程图（Graph）

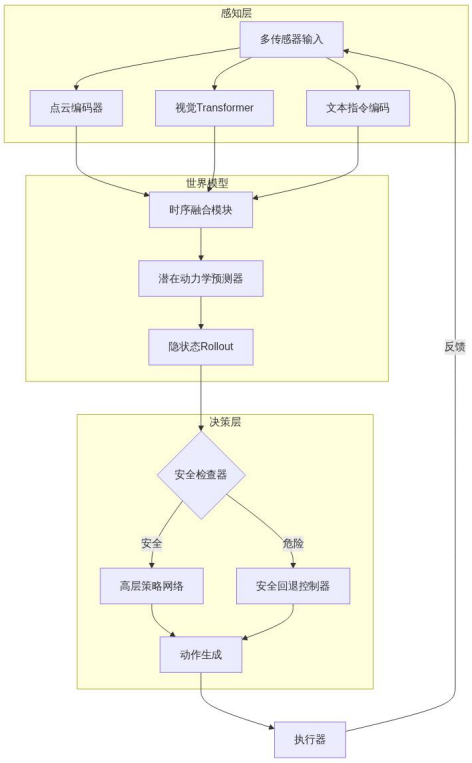

图 1: 图 1 Mermaid 图表

图 2：时序序列图（Sequence Diagram）

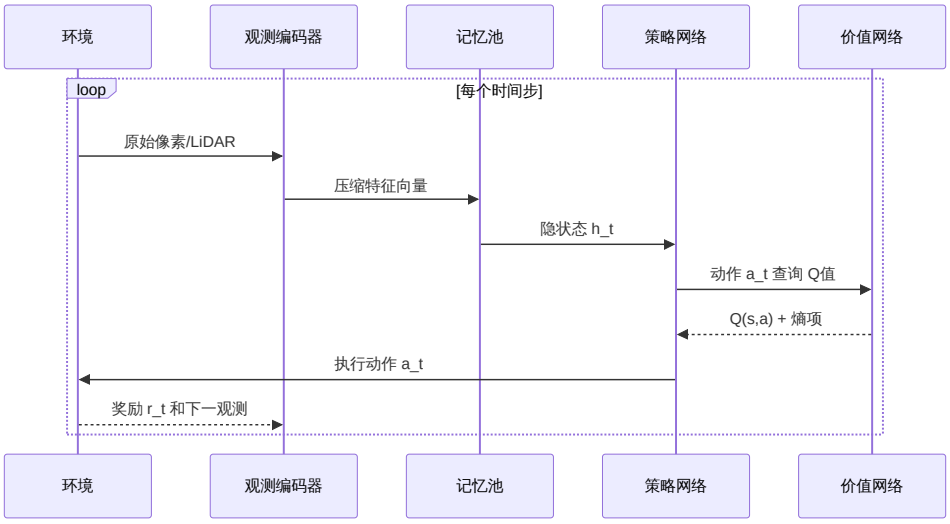

图 2: 图 2 Mermaid 图表

图 3：状态机图（State Diagram）- 智能体生命周期

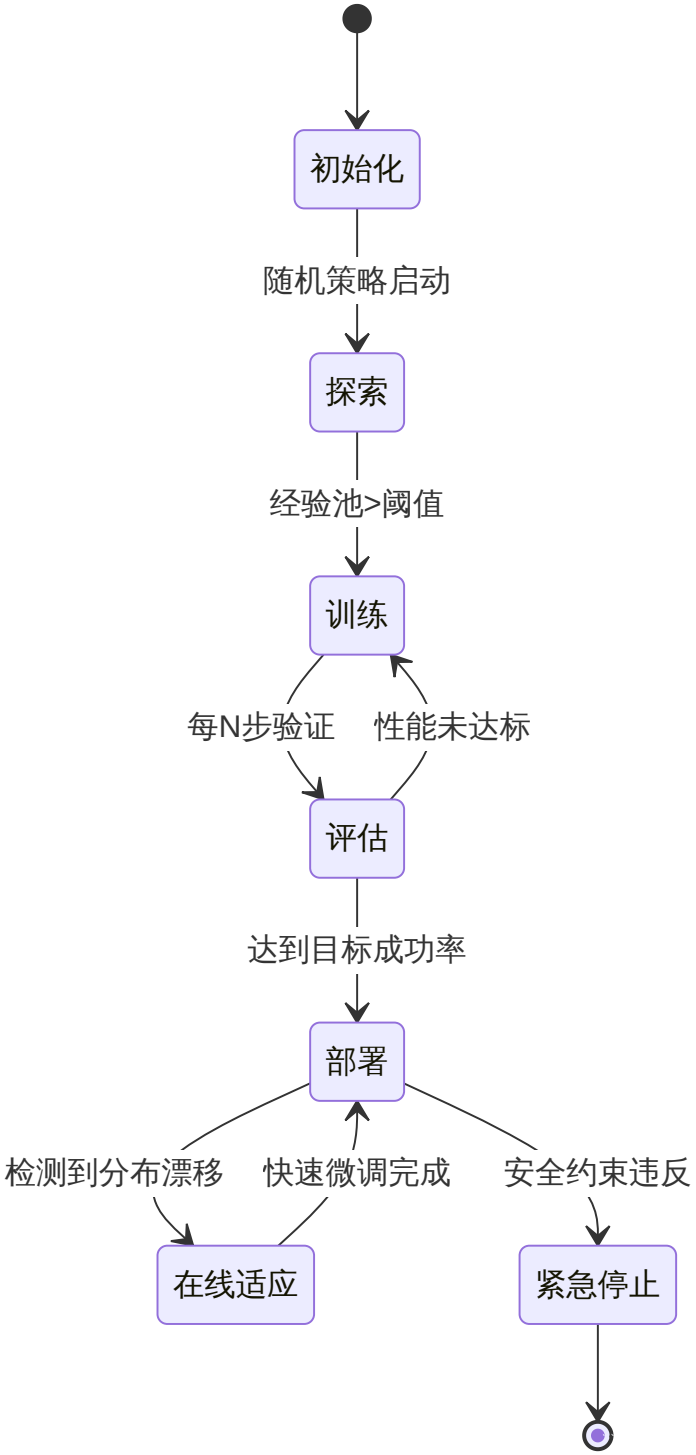

图 3: 图 3 Mermaid 图表

图 4：类图（Class Diagram）- 核心模块设计

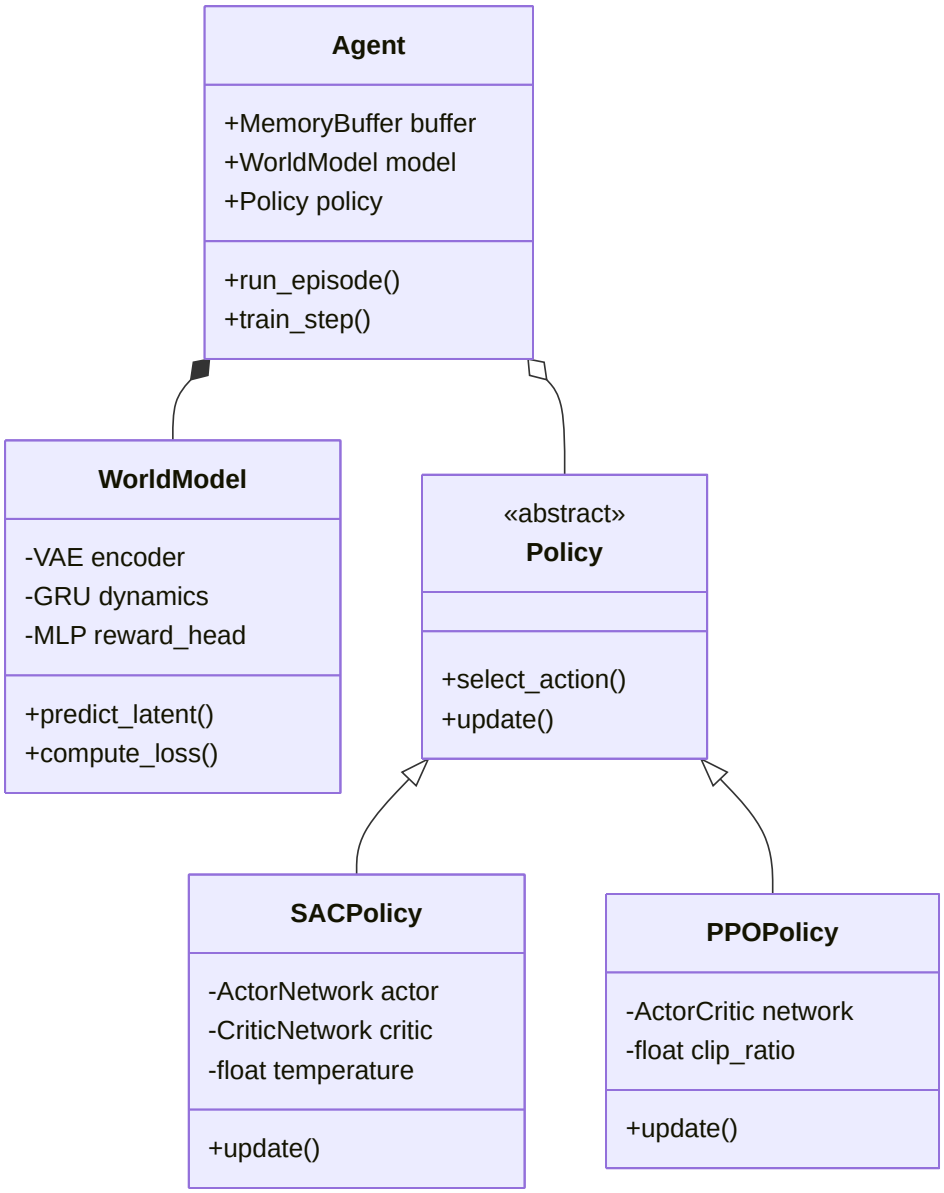

图 4: 图 4 Mermaid 图表

图 5：甘特图 - 训练任务调度

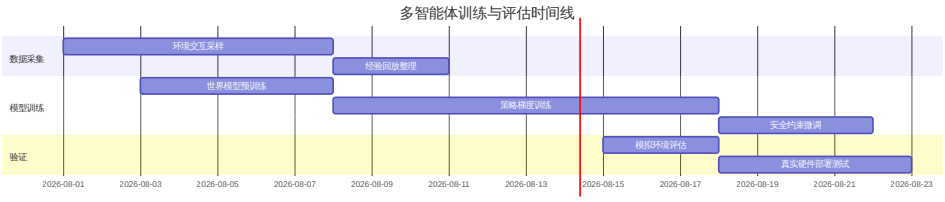

图 5: 图 5 Mermaid 图表

图 6：实体关系图（ER Diagram）- 实验数据存储

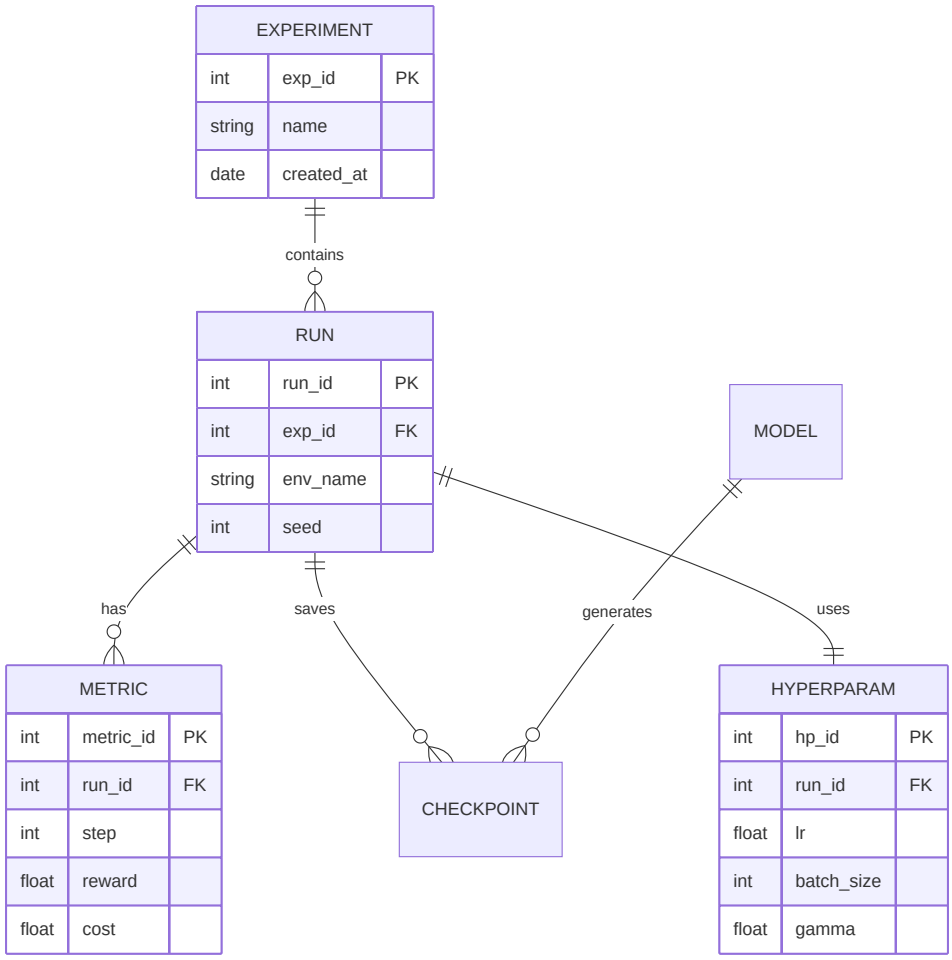

图 6: 图 6 Mermaid 图表

图 7：Git 分支模型（Git Graph）

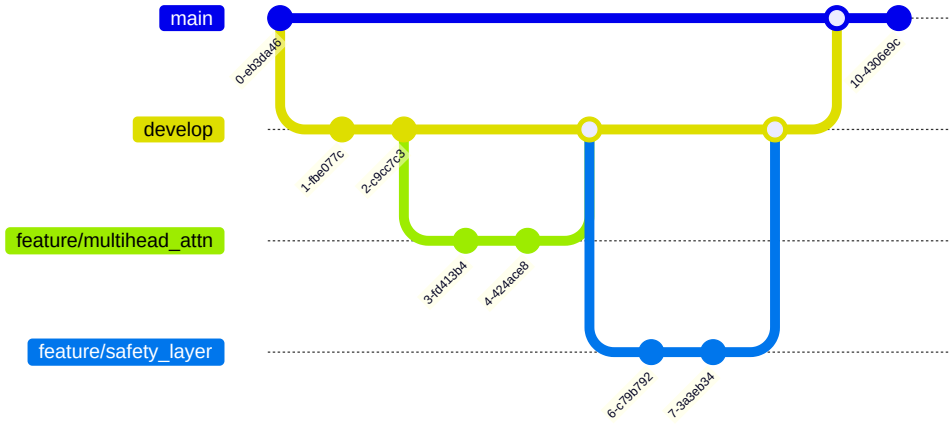

图 7: 图 7 Mermaid 图表

图 8：饼图 + 环形图组合（资源分配）

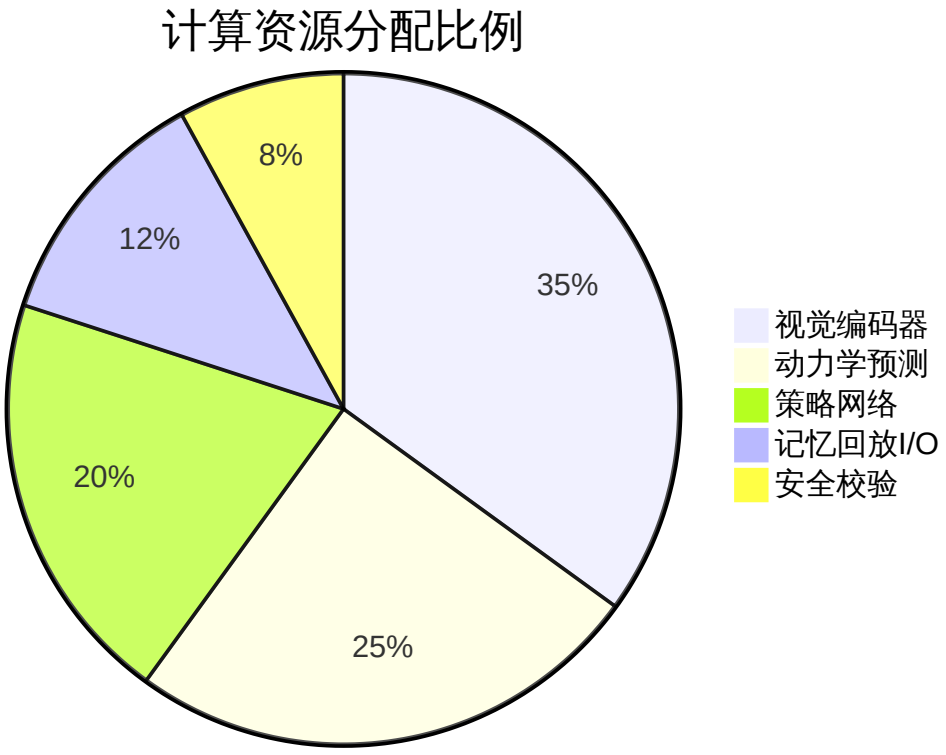

图 8: 图 8 Mermaid 图表

图 9：流程图（Flowchart）含判断 - 决策推理过程

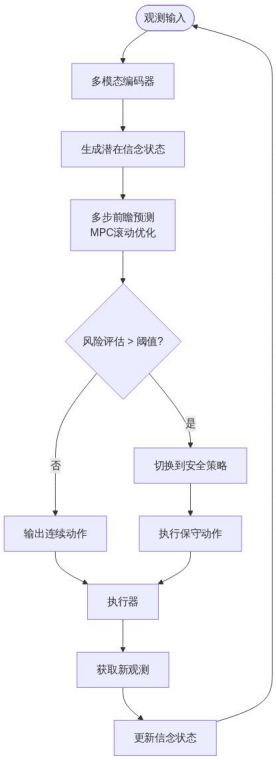

图 9: 图 9 Mermaid 图表

图 10：象限图/思维导图（Mindmap）- 损失函数族

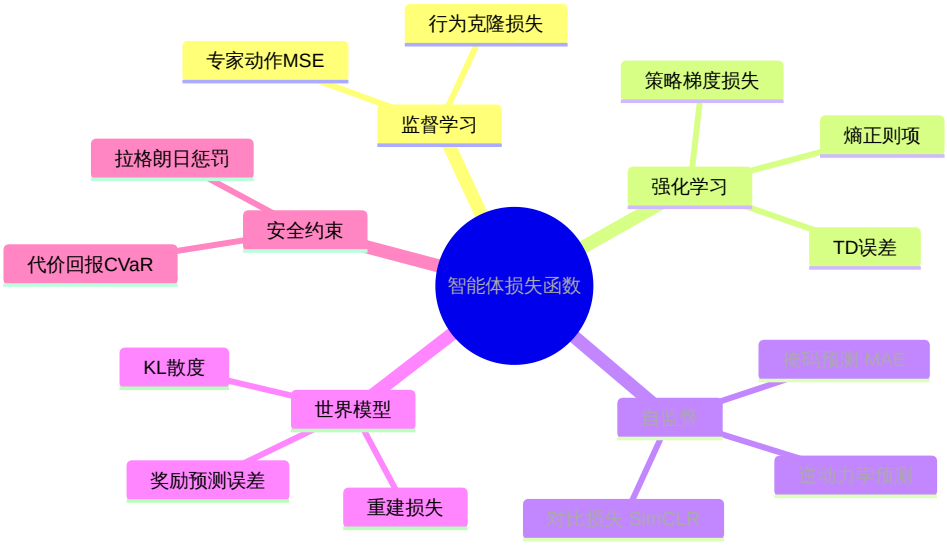

图 10: 图 10 Mermaid 图表

以上内容完整覆盖了复杂公式（含符号说明）、两张多维表格、带有类型注解和复杂逻辑的代码片段，以及 **10 种不同类型**的 Mermaid 图表（流程图、序列图、状态机、类图、甘特图、ER 图、Git 图、饼图、流程图变种、思维导图），适用于高级技术文档或论文附录。