改善版「統合アブレーション実験コード」

エコツーラボ合同会社
猪澤也寸志
polyp@webman.jp
（2025年6月10日現地時間JST）

```
# --- Cell 1.1: Google Drive Mount and Results Directory Setup (Minimized) ---
from google.colab import drive
import os

try:
    drive.mount('/content/drive', force_remount=True)
    print("✅ Google Drive mounted.")

    # ⚠️ カスタマイズ推奨（例）
    DRIVE_RESULTS_PATH = "/content/drive/MyDrive/Colab_SSG_Ablation_Study_Full_Results"

    os.makedirs(DRIVE_RESULTS_PATH, exist_ok=True)
    print(f"✅ Results directory ready: {DRIVE_RESULTS_PATH}")

except Exception as e:
    print(f"⚠️ Google Drive mount error: {e}")
    DRIVE_RESULTS_PATH = "./local_ssg_ablation_results"
    os.makedirs(DRIVE_RESULTS_PATH, exist_ok=True)
    print(f"⚠️ Using local results directory: {os.path.abspath(DRIVE_RESULTS_PATH)}")

print(f"DRIVE_RESULTS_PATH = {DRIVE_RESULTS_PATH}")
```

```
# --- セル1: Colab初回環境構築（推奨：アップデート含む）---
```

```
!pip install --upgrade datasets fsspec transformers

# 🚀 備考:
# transformers: HuggingFaceのモデル実装・Tokenizer管理
# datasets: データロード・整形ライブラリ（HuggingFace Datasets）
# fsspec: datasetsのデータI/O補助
#
# 量子PC（Qiskit, Cirqなど）を後で導入する場合はここに追記可能です。
```

```
# --- セル1（インポート確認付き）: 環境設定、ライブラリのインポート、全体設定 ---

try:
    import datasets
    print(f"✅ datasets version: {datasets.__version__}")
except ImportError as e:
    print(f"❌ datasets import failed: {e}")

try:
    import transformers
    print(f"✅ transformers version: {transformers.__version__}")
except ImportError as e:
    print(f"❌ transformers import failed: {e}")

try:
    import fsspec
    print(f"✅ fsspec version: {fsspec.__version__}")
except ImportError as e:
    print(f"❌ fsspec import failed: {e}")
```

```
# --- セル2 (完全修正版): データセットロード関数 (IMDb & SNLI対応) ---
from datasets import load_dataset
from torch.utils.data import DataLoader

def get_dataloaders(task_name, tokenizer, batch_size, num_train_samples, num_eval_samples):
    if task_name == "classification_imdb":
        print("Loading IMDb dataset from HuggingFace hub...")
        dataset = load_dataset("imdb")
    elif task_name == "nli":
        print("Loading SNLI dataset from HuggingFace hub...")
        dataset = load_dataset("snli")
    else:
        raise ValueError(f"Unknown task name: {task_name}")

    def preprocess(example):
        if task_name == "classification_imdb":
            return tokenizer(
                example["text"],
                padding="max_length",
                truncation=True,
                max_length=CFG.MAX_TOKEN_LENGTH
            )
        elif task_name == "nli":
            return tokenizer(
                example["premise"],
                example["hypothesis"],
                padding="max_length",
                truncation=True,
                max_length=CFG.MAX_TOKEN_LENGTH
            )

    print("Tokenizing dataset...")
    tokenized_dataset = dataset.map(preprocess, batched=True)

    if task_name == "classification_imdb":
        train_dataset = tokenized_dataset["train"].select(range(min(num_train_samples,
len(tokenized_dataset["train"]))))
        eval_dataset = tokenized_dataset["test"].select(range(min(num_eval_samples,
len(tokenized_dataset["test"]))))
```

```
        num_labels = 2
    elif task_name == "nli":
        train_dataset = tokenized_dataset["train"].filter(lambda x: x['label'] != -1)
        train_dataset = train_dataset.select(range(min(num_train_samples, len(train_dataset))))
        eval_dataset = tokenized_dataset["validation"].filter(lambda x: x['label'] != -1)
        eval_dataset = eval_dataset.select(range(min(num_eval_samples, len(eval_dataset))))
        num_labels = 3

    # Rename 'label' -> 'labels' for consistency with training pipeline
    train_dataset = train_dataset.rename_column("label", "labels")
    eval_dataset = eval_dataset.rename_column("label", "labels")

    # Set dataset format
    columns = ['input_ids', 'attention_mask', 'labels']
    train_dataset.set_format(type='torch', columns=columns)
    eval_dataset.set_format(type='torch', columns=columns)

    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
    eval_loader = DataLoader(eval_dataset, batch_size=batch_size)

    return train_loader, eval_loader, num_labels

print("--- Cell 2: Dataset Loader and Preprocessor (Fixed) ready. ---")
```

```
# --- セル1 (完全修正版): 環境設定、ライブラリのインポート、全体設定 ---

# 標準ライブラリ
import os
import json
import time
import math
import copy
import io
```

```
import random
import numpy as np
import pandas as pd
from tqdm.auto import tqdm
import tarfile
import zipfile
import urllib.request

# PyTorch系
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader

# HuggingFace Transformers系
from datasets import load_dataset, Dataset, DatasetDict
from transformers import (
    AutoConfig,
    AutoModel,
    AutoTokenizer,
    get_linear_schedule_with_warmup,
)
from sklearn.metrics import accuracy_score

# --- 実験設定 (GlobalConfig) ---
class GlobalConfig:
    SEED = 42
    BASE_MODEL_NAME = 'bert-base-uncased'
    DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    TRAIN_SAMPLES = 10   # 実験デバッグ用
    EVAL_SAMPLES = 2     # 実験デバッグ用
    MAX_TOKEN_LENGTH = 256
    EPOCHS = 2
    BATCH_SIZE = 16
    LEARNING_RATE = 2e-5

# 乱数シード固定 (再現性)
def set_seed(seed_value):
    random.seed(seed_value)
    np.random.seed(seed_value)
    torch.manual_seed(seed_value)
```

```
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed_value)

CFG = GlobalConfig()
set_seed(CFG.SEED)

# --- 実行環境の確認 ---
print(f"--- Environment Setup ---")
print(f"PyTorch Version: {torch.__version__}")
print(f"Device: {CFG.DEVICE}")
print(f"Base Model: {CFG.BASE_MODEL_NAME}")
print(f"Seed: {CFG.SEED}")
print(f"-------------------------")









# --- セル2: データセットのロードとDataLoader準備 ---

from datasets import load_dataset
from torch.utils.data import DataLoader

def load_and_prepare_dataset(task_name, tokenizer, train_samples, eval_samples, max_length):
    """
    IMDbまたはSNLIのデータセットをロードし、トークナイズしてDataLoaderを返す関数。
    """
    if task_name == 'classification_imdb':
        print("Loading IMDb dataset from HuggingFace hub...")
        raw_datasets = load_dataset('imdb')
        train_dataset = raw_datasets['train']×shuffle(seed=CFG.SEED).select(range(train_samples))
        eval_dataset = raw_datasets['test']×shuffle(seed=CFG.SEED).select(range(eval_samples))

        def tokenize_function(examples):
            return tokenizer(examples['text'], padding='max_length', truncation=True,
max_length=max_length)
```

```
        num_labels = 2

    elif task_name == 'nli':
        print("Loading SNLI dataset from HuggingFace hub...")
        raw_datasets = load_dataset('snli')
        train_dataset = raw_datasets['train'].filter(lambda x: x['label'] !=
-1)×shuffle(seed=CFG.SEED).select(range(train_samples))
        eval_dataset = raw_datasets['validation'].filter(lambda x: x['label'] !=
-1)×shuffle(seed=CFG.SEED).select(range(eval_samples))

        def tokenize_function(examples):
            return tokenizer(
                examples['premise'],
                examples['hypothesis'],
                padding='max_length',
                truncation=True,
                max_length=max_length
            )

        num_labels = 3

    else:
        raise ValueError(f"Unknown task_name: {task_name}")

    # Tokenize datasets
    train_dataset = train_dataset.map(tokenize_function, batched=True)
    eval_dataset = eval_dataset.map(tokenize_function, batched=True)

    # Rename 'label' -> 'labels' for consistency
    train_dataset = train_dataset.rename_column('label', 'labels')
    eval_dataset = eval_dataset.rename_column('label', 'labels')

    # Set dataset format
    train_dataset.set_format(type='torch', columns=['input_ids', 'attention_mask', 'labels'])
    eval_dataset.set_format(type='torch', columns=['input_ids', 'attention_mask', 'labels'])

    train_loader = DataLoader(train_dataset, batch_size=CFG.BATCH_SIZE, shuffle=True)
    eval_loader = DataLoader(eval_dataset, batch_size=CFG.BATCH_SIZE, shuffle=False)

    return train_loader, eval_loader, num_labels
```

```
print("--- Cell 2: Dataset Loader and Preprocessor ready. ---")
```

```
# --- セル3: PipelineModel（SSG統合対応）---

class PipelineModel(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.base_model_name = config['base_model_name']
        self.num_labels = config['num_labels']
        self.enabled_modules = config.get('enabled_modules', [])

        # Transformer Backbone
        self.base_model = AutoModel.from_pretrained(self.base_model_name)
        base_hidden_dim = self.base_model.config.hidden_size

        # 追加モジュール
        self.modules_list = nn.ModuleList()
        for module_name in self.enabled_modules:
            if module_name == 'semantic_attention':
                self.modules_list.append(SemanticAttentionModule(base_hidden_dim))
            elif module_name == 'semantic_concept':
                self.modules_list.append(SemanticConceptModule(base_hidden_dim))
            # 他のモジュールはここに追加できます
            else:
                print(f"⚠️ Warning: Module '{module_name}' is not recognized and will be skipped.")

        # 分類ヘッド
        self.classifier = nn.Linear(base_hidden_dim, self.num_labels)

    def forward(self, input_ids, attention_mask, labels=None):
        outputs = self.base_model(input_ids=input_ids, attention_mask=attention_mask)
        hidden_states = outputs.last_hidden_state

        # SSGモジュール処理（逐次）
        for module in self.modules_list:
```

```
            module_output = module(hidden_states, attention_mask)
            hidden_states = module_output['last_hidden_state']

        # プーリング (平均プーリング)
        pooled_output = hidden_states.mean(dim=1)

        # 分類
        logits = self.classifier(pooled_output)

        # 損失計算 (訓練時)
        loss = None
        if labels is not None:
            loss = F.cross_entropy(logits, labels)

        return {'loss': loss, 'logits': logits}

print("--- Cell 3: PipelineModel (SSG対応) defined. ---")
```

```
# --- Cell 3b: PipelineModel (Notebook対応: モジュール直接参照) ---
import torch
import torch.nn as nn
from transformers import AutoModel

class PipelineModel(nn.Module):
    def __init__(self, config):
        super(PipelineModel, self).__init__()
        base_model_name = config["base_model_name"]
        num_labels = config["num_labels"]
        enabled_modules = config×get("enabled_modules", [])

        self.base_model = AutoModel.from_pretrained(base_model_name)
        hidden_size = self.base_model.config.hidden_size

        self.modules_dict = nn.ModuleDict()
```

```
        # SemanticAttentionModule
        if "semantic_attention" in enabled_modules:
            self.modules_dict["semantic_attention"] = SemanticAttentionModule(
                input_dim=hidden_size, attention_dim=64
            )

        # SemanticConceptModule
        if "semantic_concept" in enabled_modules:
            self.modules_dict["semantic_concept"] = SemanticConceptModule(
                input_dim=hidden_size
            )

        # DirectNonCommutativeModule
        if "direct_non_commutative" in enabled_modules:
            self.modules_dict["direct_non_commutative"] = DirectNonCommutativeModule(
                input_dim=hidden_size
            )

        # 最終出力層
        self.classifier = nn.Linear(hidden_size, num_labels)

    def forward(self, input_ids, attention_mask):
        outputs = self.base_model(input_ids=input_ids, attention_mask=attention_mask)
        hidden_states = outputs.last_hidden_state  # (batch, seq_len, hidden_dim)

        for module_name, module in self.modules_dict.items():
            result = module(hidden_states)
            hidden_states = result["last_hidden_state"]

        pooled_output = hidden_states[:, 0, :]  # [CLS] token
        logits = self.classifier(pooled_output)
        return {"logits": logits}

print("--- Cell 3b: PipelineModel (Notebook対応版) defined. ---")
```

```
# --- Cell 4: SemanticAttentionModule (attention_dim対応版) ---
import torch
import torch.nn as nn

class SemanticAttentionModule(nn.Module):
    def __init__(self, input_dim, attention_dim=64, model_config=None,
module_specific_config=None):
        super(SemanticAttentionModule, self).__init__()
        self.input_dim = input_dim
        self.attention_dim = attention_dim

        # attention_dim の取得
        if module_specific_config is not None and 'attention_dim' in module_specific_config:
            self.attention_dim = module_specific_config['attention_dim']

        # Attention層
        self.attention_layer = nn.Sequential(
            nn.Linear(self.input_dim, self.attention_dim),
            nn.Tanh(),
            nn.Linear(self.attention_dim, 1)
        )

        print(f"SemanticAttentionModule initialized: input_dim={self.input_dim},
attention_dim={self.attention_dim}")

    def forward(self, hidden_states, attention_mask=None, **kwargs):
        # hidden_states: (batch_size, seq_len, input_dim)
        attention_scores = self.attention_layer(hidden_states).squeeze(-1)  # (batch_size, seq_len)
        if attention_mask is not None:
            attention_scores = attention_scores.masked_fill(attention_mask == 0, -1e9)
        attention_weights = torch.softmax(attention_scores, dim=1).unsqueeze(-1)  # (batch_size,
seq_len, 1)

        # 重み付け和
        context_vector = torch.sum(hidden_states * attention_weights, dim=1, keepdim=True)
        output = context_vector.repeat(1, hidden_states.size(1), 1)  # (batch_size, seq_len,
input_dim)
```

```
        return {"last_hidden_state": output, "attention_weights": attention_weights.detach()}

    def get_output_dim(self):
        return self.input_dim

print("--- Cell 4: SemanticAttentionModule (attention_dim対応版) defined. ---")









# --- セル5: SemanticConceptModule（意味概念モジュール）---

class SemanticConceptModule(nn.Module):
    def __init__(self, input_dim, module_specific_config=None):
        super().__init__()
        self.input_dim = input_dim

        cfg = module_specific_config if module_specific_config is not None else {}
        self.num_concepts = cfg.get("num_concepts", 32)
        self.concept_dim = cfg.get("concept_dim", 128)

        # 概念プロトタイプ
        self.concept_prototypes = nn.Parameter(torch.randn(self.num_concepts,
self.concept_dim))
        nn.init.xavier_uniform_(self.concept_prototypes)

        self.hidden_to_concept_proj = nn.Linear(self.input_dim, self.concept_dim)
        self.concepts_to_integrate_proj = nn.Linear(self.concept_dim, self.input_dim)
        self.activation = nn.ReLU()

        print(f"SemanticConceptModule initialized: input_dim={self.input_dim}, "
              f"num_concepts={self.num_concepts}, concept_dim={self.concept_dim}")

    def forward(self, hidden_states, attention_mask=None, **kwargs):
        # hidden_states: (batch_size, seq_len, input_dim)

        projected_hidden = self×activation(self×hidden_to_concept_proj(hidden_states))
```

```
        # projected_hidden: (batch_size, seq_len, concept_dim)

        attention_scores = torch.matmul(projected_hidden, self.concept_prototypes.t())
        # attention_scores: (batch_size, seq_len, num_concepts)

        if attention_mask is not None:
            expanded_attention_mask = attention_mask.unsqueeze(-1).expand_as(attention_scores)
            attention_scores = attention_scores.masked_fill(expanded_attention_mask == 0, -1e9)

        attention_weights = torch.softmax(attention_scores, dim=-1)
        # attention_weights: (batch_size, seq_len, num_concepts)

        contextual_concepts = torch.matmul(attention_weights, self.concept_prototypes)
        # contextual_concepts: (batch_size, seq_len, concept_dim)

        projected_contextual_concepts = self.activation(
            self.concepts_to_integrate_proj(contextual_concepts)
        )
        # projected_contextual_concepts: (batch_size, seq_len, input_dim)

        fused_hidden_states = hidden_states + projected_contextual_concepts
        # 出力 (hidden_states と概念情報の融合)

        return {"last_hidden_state": fused_hidden_states, "attention_weights":
attention_weights.detach()}

    def get_output_dim(self):
        return self.input_dim

print("--- Cell 5: SemanticConceptModule defined. ---")




# --- Cell 6: DirectNonCommutativeModule (Optimized) ---

import torch
```

```
import torch.nn as nn

class DirectNonCommutativeModule(nn.Module):
    def __init__(self, input_dim, output_dim=None, model_config=None,
module_specific_config=None):
        super().__init__()
        self.input_dim = input_dim

        # 設定取得
        cfg = module_specific_config if module_specific_config is not None else {}
        self.verbose = cfg.get("verbose", True)
        self.internal_dropout = cfg.get("dropout", 0.2)
        # デフォルト：input_dimの2倍（拡張性あり）
        self.internal_ff_dim = cfg.get("internal_processing_dim", input_dim * 2)

        # 出力次元
        self.output_dim = output_dim or cfg.get("output_dim", input_dim)

        # 非可換プロセッサ（順序反転差分）
        self.non_commutative_processor = nn.Sequential(
            nn.Linear(input_dim * 2, self.internal_ff_dim),
            nn.LayerNorm(self.internal_ff_dim),
            nn.ReLU(),
            nn.Dropout(self.internal_dropout),
            nn.Linear(self.internal_ff_dim, input_dim)
        )

        # λパラメータ（学習可能）
        self.lambda_param = nn.Parameter(torch.tensor(cfg.get("lambda_init", 0.5)))

        # 出力射影（出力次元が異なる場合）
        if self.input_dim != self.output_dim:
            self.final_integration_projection = nn.Linear(input_dim, self.output_dim)
        else:
            self.final_integration_projection = None

        # ログ
        if self.verbose:
            print(f"DirectNonCommutativeModule initialized: "
                  f"input_dim={self.input_dim}, output_dim={self.output_dim}, "
                  f"internal_ff_dim={self.internal_ff_dim}, dropout={self.internal_dropout}, "
```

```
            f"lambda_init={self.lambda_param.item()}")

def forward(self, hidden_states, attention_mask=None, **kwargs):
    """
    hidden_states: (batch_size, seq_len, input_dim)
    """
    batch_size, seq_len, _ = hidden_states.shape
    device = hidden_states.device

    # シーケンス長が2未満の場合（安全措置）
    if seq_len < 2:
        output_features = (
            self.final_integration_projection(hidden_states)
            if self.final_integration_projection
            else hidden_states
        )
        return {
            "last_hidden_state": output_features,
            "non_commutative_effects_map": None
        }

    # 非可換効果格納
    non_commutative_effects_at_each_step = torch.zeros_like(hidden_states)

    for i in range(seq_len - 1):
        h_i = hidden_states[:, i, :]
        h_next = hidden_states[:, i + 1, :]

        forward_concat = torch×cat((h_i, h_next), dim=-1)
        backward_concat = torch×cat((h_next, h_i), dim=-1)

        forward_processed = self.non_commutative_processor(forward_concat)
        backward_processed = self.non_commutative_processor(backward_concat)

        # λでスケーリング
        non_comm_effect = self.lambda_param * (forward_processed - backward_processed)

        # 両トークンに半分ずつ分配
        non_commutative_effects_at_each_step[:, i, :] += 0.5 * non_comm_effect
        non_commutative_effects_at_each_step[:, i + 1, :] -= 0.5 * non_comm_effect
```

```
        fused_hidden_states = hidden_states + non_commutative_effects_at_each_step

        output_features = (
            self.final_integration_projection(fused_hidden_states)
            if self.final_integration_projection
            else fused_hidden_states
        )

        # 出力: last_hidden_stateと非可換効果マップ
        return {
            "last_hidden_state": output_features,
            "non_commutative_effects_map": non_commutative_effects_at_each_step.detach()
        }

    def get_output_dim(self):
        return self.output_dim

print("--- Cell 6: DirectNonCommutativeModule (Optimized) defined. ---")




# --- Cell 7: DiscontinuityEmergenceModule class (Optimized for Current Experiment Code) ---
import torch
import torch.nn as nn

class DiscontinuityEmergenceModule(nn.Module):
    def __init__(self, input_dim, output_dim=None, model_config=None,
module_specific_config=None):
        super().__init__()
        self.input_dim = input_dim
        cfg = module_specific_config if module_specific_config is not None else {}

        self.semantic_dim_for_discontinuity = cfg×get("semantic_dim_for_discontinuity", 12)
        self.emergent_label_dim = cfg.get("emergent_label_dim", 24)
        internal_dropout = cfg.get("dropout", 0.2)
```

```
        # 出力次元設定（入力＋創発ラベル次元 or 指定）
        potential_intermediate_dim = self.input_dim + self.emergent_label_dim
        self.output_dim = cfg×get("output_dim") or output_dim or potential_intermediate_dim

        # 断絶検出のための意味特徴抽出
        self.feature_to_semantic = nn.Sequential(
            nn.Linear(self.input_dim, self.input_dim // 2),
            nn.ReLU(),
            nn.Linear(self.input_dim // 2, self.semantic_dim_for_discontinuity),
            nn.Tanh()
        )

        # 断絶スコア検出器
        self.discontinuity_detector = nn.Sequential(
            nn.Linear(self.semantic_dim_for_discontinuity * 2, 64),
            nn.ReLU(),
            nn.Dropout(internal_dropout),
            nn.Linear(64, 1),
            nn.Sigmoid()
        )

        # 創発ラベル生成器
        self.emergent_label_generator = nn.Sequential(
            nn.Linear(self.semantic_dim_for_discontinuity * 2, self.input_dim // 4),
            nn.LayerNorm(self.input_dim // 4),
            nn.ReLU(),
            nn.Dropout(internal_dropout),
            nn.Linear(self.input_dim // 4, self.emergent_label_dim),
            nn.Tanh()
        )

        # 出力次元調整用projection（必要に応じて）
        if potential_intermediate_dim != self.output_dim:
            self.integration_projection = nn.Linear(potential_intermediate_dim, self.output_dim)
        elif self.input_dim != self.output_dim:
            self.fallback_projection = nn.Linear(self.input_dim, self.output_dim)

        print(f"[DiscontinuityEmergenceModule] Initialized "
              f"(input_dim={self.input_dim}, output_dim={self.output_dim}, "
              f"semantic_dim={self.semantic_dim_for_discontinuity},
emergent_dim={self.emergent_label_dim})")
```

```
    def forward(self, hidden_states, attention_mask=None, **kwargs):
        batch_size, seq_len, _ = hidden_states.shape
        device = hidden_states.device

        semantic_features = self.feature_to_semantic(hidden_states)

        disc_scores_seq = torch.zeros(batch_size, max(seq_len - 1, 1), 1, device=device)
        emergent_labels_seq = torch.zeros(batch_size, max(seq_len - 1, 1),
self.emergent_label_dim, device=device)

        if seq_len > 1:
            sfd_t = semantic_features[:, :-1, :]
            sfd_tp1 = semantic_features[:, 1:, :]
            combined = torch.cat((sfd_t, sfd_tp1), dim=-1)

            disc_scores_seq = self.discontinuity_detector(combined)
            emergent_labels_seq = self.emergent_label_generator(combined)

        # pooled_emergent_label: attention_mask考慮で平均 (or mean)
        if emergent_labels_seq.shape[1] > 0:
            if attention_mask is not None and attention_mask.shape[1] > 1:
                mask = attention_mask[:, 1:].unsqueeze(-1)
                pooled_label = (emergent_labels_seq × mask)×sum(dim=1) /
torch×clamp(mask×sum(dim=1), min=1)
            else:
                pooled_label = emergent_labels_seq×mean(dim=1)

            expanded_label = pooled_label.unsqueeze(1).expand(-1, seq_len, -1)
            output_features = torch.cat((hidden_states, expanded_label), dim=-1)
        else:
            if self.output_dim == self.input_dim + self.emergent_label_dim:
                zero_label = torch.zeros(batch_size, seq_len, self.emergent_label_dim, device=device)
                output_features = torch.cat((hidden_states, zero_label), dim=-1)
            else:
                output_features = hidden_states

        if hasattr(self, 'integration_projection'):
            final_output = self.integration_projection(output_features)
        elif hasattr(self, 'fallback_projection'):
            final_output = self.fallback_projection(output_features)
```

```
        else:
            final_output = output_features

        return {
            "last_hidden_state": final_output,
            "discontinuity_scores": disc_scores_seq.squeeze(-1).detach(),
            "emergent_labels_generated": emergent_labels_seq.detach()
        }

    def get_output_dim(self):
        return self.output_dim

print("--- Cell 7: DiscontinuityEmergenceModule class defined (Optimized). ---")









# --- Cell 8: InterpretableHolographicReducer class (Optimized) ---
import torch
import torch.nn as nn
import torch.nn.functional as F

class InterpretableHolographicReducer(nn.Module):
    """
    InterpretableHolographicReducer
    本モジュールは、入力特徴量から解釈可能なコンセプトごとに特徴縮減を行い、
    文脈的に解釈可能な多次元特徴表現を生成するためのモジュールです。
    """

    def __init__(self, input_feature_dim, output_dim=None, model_config=None,
module_specific_config=None):
        """
        Args:
            input_feature_dim (int): 入力特徴量の次元数
            output_dim (int, optional): 最終出力次元。Noneの場合は num_key_concepts *
reduced_dim_per_concept を使用
```

```
        model_config (dict, optional): 全体モデルの設定辞書（未使用、互換性用）
        module_specific_config (dict, optional): このモジュール固有の設定辞書
    """
    super().__init__()
    self.input_feature_dim = input_feature_dim

    cfg = module_specific_config if module_specific_config is not None else {}
    self.num_key_concepts = cfg.get("num_key_concepts", 16)
    self.reduced_dim_per_concept = cfg.get("reduced_dim_per_concept", 32)  # 0601PDFで調
整

    # コンセプトプロトタイプ（学習可能パラメータ）
    self.key_concept_prototypes = nn.Parameter(
        torch.empty(self.num_key_concepts, self.input_feature_dim)
    )
    nn.init.xavier_uniform_(self.key_concept_prototypes)

    # 入力特徴量を各コンセプトベクトルに射影
    self.reduction_projector = nn.Linear(self.input_feature_dim, self.reduced_dim_per_concept)
    self.activation_after_projection = nn.Tanh()

    # 出力次元の扱い
    default_output_dim = self.num_key_concepts * self.reduced_dim_per_concept
    self.output_dim = output_dim if output_dim is not None else default_output_dim

    print(f"InterpretableHolographicReducer initialized: "
          f"input_dim={self.input_feature_dim}, "
          f"num_key_concepts={self.num_key_concepts}, "
          f"reduced_dim_per_concept={self.reduced_dim_per_concept}, "
          f"output_dim={self.output_dim}")

def forward(self, hidden_states_sequence, attention_mask=None, **kwargs):
    """
    Args:
        hidden_states_sequence (Tensor): (batch_size, seq_len, input_feature_dim)
        attention_mask (Tensor, optional): (batch_size, seq_len)
    Returns:
        dict: {
            "last_hidden_state": (batch_size, output_dim),
            "concept_attention_map": (batch_size, num_key_concepts, seq_len) or None
        }
```

```
        """
        batch_size, seq_len, _ = hidden_states_sequence.shape
        reduced_concept_vectors = []
        all_attention_weights = []

        for i in range(self.num_key_concepts):
            concept_proto = self.key_concept_prototypes[i]  # (input_feature_dim)

            # 各トークンと概念プロトタイプの類似度スコア
            scores = torch×matmul(hidden_states_sequence,
concept_proto.unsqueeze(-1)).squeeze(-1)

            if attention_mask is not None:
                scores = scores×masked_fill(attention_mask == 0, -1e9)

            attn_weights = F.softmax(scores, dim=1)
            all_attention_weights.append(attn_weights.unsqueeze(1))  # (batch_size, 1, seq_len)

            # 重み付け平均 (文脈表現)
            context_vector = torch×sum(
                hidden_states_sequence * attn_weights.unsqueeze(-1), dim=1
            )

            # 射影＋活性化
            reduced_vector = self.activation_after_projection(
                self.reduction_projector(context_vector)
            )
            reduced_concept_vectors.append(reduced_vector)  # (batch_size,
reduced_dim_per_concept)

        # コンセプトごとの縮減ベクトルを連結
        final_representation = torch.cat(reduced_concept_vectors, dim=1)  # (batch_size,
output_dim)

        concept_attention_map = torch.cat(all_attention_weights, dim=1) if all_attention_weights
else None

        return {
            "last_hidden_state": final_representation,
            "concept_attention_map": concept_attention_map.detach() if concept_attention_map is
not None else None
```

```
        }

    def get_output_dim(self):
        """
        Returns:
            int: モジュールの出力次元数
        """
        return self.output_dim

print("--- Cell 8: InterpretableHolographicReducer class defined. ---")
```

```
# --- Cell 7: Module Registry Cell (Full Extended Version) ---

# IMDb
experiments_to_run = [
    {
        "Experiment_Name": "E0_Baseline_IMDb",
        "Task_Type": "classification_imdb",
        "Enabled_Modules": [],
    },
    {
        "Experiment_Name": "E1_SemAtt_IMDb",
        "Task_Type": "classification_imdb",
        "Enabled_Modules": ["semantic_attention"],
    },
    {
        "Experiment_Name": "E2_SemConcept_IMDb",
        "Task_Type": "classification_imdb",
        "Enabled_Modules": ["semantic_concept"],
    },
    {
        "Experiment_Name": "E3_NonComm_IMDb",
        "Task_Type": "classification_imdb",
        "Enabled_Modules": ["direct_non_commutative"],
```

```
    },
    {
        "Experiment_Name": "E4_DiscEmergence_IMDb",
        "Task_Type": "classification_imdb",
        "Enabled_Modules": ["discontinuity_emergence"],
    },
    {
        "Experiment_Name": "E5_HoloReducer_IMDb",
        "Task_Type": "classification_imdb",
        "Enabled_Modules": ["interpretable_holographic_reducer"],
    },
    {
        "Experiment_Name": "E6_AllModules_IMDb",
        "Task_Type": "classification_imdb",
        "Enabled_Modules": [
            "semantic_attention",
            "semantic_concept",
            "direct_non_commutative",
            "discontinuity_emergence",
            "interpretable_holographic_reducer"
        ],
        "Notes": "All modules combined for IMDb"
    },
]

# SNLI
experiments_to_run += [
    {
        "Experiment_Name": "E0_Baseline_NLI",
        "Task_Type": "nli",
        "Enabled_Modules": [],
    },
    {
        "Experiment_Name": "E1_SemAtt_NLI",
        "Task_Type": "nli",
        "Enabled_Modules": ["semantic_attention"],
    },
    {
        "Experiment_Name": "E2_SemConcept_NLI",
        "Task_Type": "nli",
        "Enabled_Modules": ["semantic_concept"],
```

```
    },
    {
        "Experiment_Name": "E3_NonComm_NLI",
        "Task_Type": "nli",
        "Enabled_Modules": ["direct_non_commutative"],
    },
    {
        "Experiment_Name": "E4_DiscEmergence_NLI",
        "Task_Type": "nli",
        "Enabled_Modules": ["discontinuity_emergence"],
    },
    {
        "Experiment_Name": "E5_HoloReducer_NLI",
        "Task_Type": "nli",
        "Enabled_Modules": ["interpretable_holographic_reducer"],
    },
    {
        "Experiment_Name": "E6_AllModules_NLI",
        "Task_Type": "nli",
        "Enabled_Modules": [
            "semantic_attention",
            "semantic_concept",
            "direct_non_commutative",
            "discontinuity_emergence",
            "interpretable_holographic_reducer"
        ],
        "Notes": "All modules combined for SNLI"
    },
]

print("--- Cell 7: Module Registry Cell (Full Extended Version) defined. ---")




# --- Cell 7b: Experiment Runner (Optimized Final Version) ---
import torch
import torch.nn.functional as F
```

```
from tqdm.auto import tqdm

def train_one_epoch(model, dataloader, optimizer, scheduler=None):
    """
    1エポックの学習を行う関数。
    Args:
        model: PipelineModel
        dataloader: DataLoader
        optimizer: torch.optim
        scheduler: 学習率スケジューラー（任意）
    Returns:
        tuple: (平均損失, 平均正解率)
    """
    model.train()
    total_loss, total_correct, total_samples = 0.0, 0, 0

    for batch in dataloader:
        input_ids = batch["input_ids"].to(CFG.DEVICE)
        attention_mask = batch["attention_mask"].to(CFG.DEVICE)
        labels = batch["labels"].to(CFG.DEVICE)

        optimizer.zero_grad()
        outputs = model(input_ids, attention_mask)
        logits = outputs["logits"]
        loss = F.cross_entropy(logits, labels)

        loss.backward()
        optimizer.step()
        if scheduler is not None:
            scheduler.step()

        total_loss += loss.item() * labels.size(0)
        total_correct += (logits×argmax(dim=1) == labels).sum().item()
        total_samples += labels.size(0)

    avg_loss = total_loss / total_samples if total_samples > 0 else float('inf')
    avg_acc = total_correct / total_samples if total_samples > 0 else 0.0
    return avg_loss, avg_acc

def evaluate_one_epoch(model, dataloader):
    """
```

```
    1エポックの評価を行う関数。
    Args:
        model: PipelineModel
        dataloader: DataLoader
    Returns:
        tuple: (平均損失, 平均正解率)
    """
    model.eval()
    total_loss, total_correct, total_samples = 0.0, 0, 0

    with torch.no_grad():
        for batch in dataloader:
            input_ids = batch["input_ids"].to(CFG.DEVICE)
            attention_mask = batch["attention_mask"].to(CFG.DEVICE)
            labels = batch["labels"].to(CFG.DEVICE)

            outputs = model(input_ids, attention_mask)
            logits = outputs["logits"]
            loss = F.cross_entropy(logits, labels)

            total_loss += loss.item() * labels.size(0)
            total_correct += (logits×argmax(dim=1) == labels).sum().item()
            total_samples += labels.size(0)

    avg_loss = total_loss / total_samples if total_samples > 0 else float('inf')
    avg_acc = total_correct / total_samples if total_samples > 0 else 0.0
    return avg_loss, avg_acc

def run_experiment(exp_config, train_loader, eval_loader, num_labels):
    """
    実験設定に基づきPipelineModelを構築し、学習と評価を行う関数。
    Args:
        exp_config (dict): 実験設定
        train_loader (DataLoader): 訓練データ
        eval_loader (DataLoader): 評価データ
        num_labels (int): 分類ラベル数
    """
    experiment_name = exp_config.get('Experiment_Name', 'Unnamed_Experiment')
    print(f"==================== Running Experiment: {experiment_name}
====================")
```

```
    try:
        # モデル構築
        model_config = {
            "base_model_name": CFG.BASE_MODEL_NAME,
            "num_labels": num_labels,
            "enabled_modules": exp_config.get("Enabled_Modules", [])
        }
        model = PipelineModel(model_config)
        model.to(CFG.DEVICE)

        # オプティマイザとスケジューラ
        optimizer = torch.optim.AdamW(model.parameters(), lr=CFG.LEARNING_RATE)
        total_steps = len(train_loader) * CFG.EPOCHS
        scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=1, gamma=0.9)

        print(f"Enabled Modules: {exp_config.get('Enabled_Modules', [])}")

        for epoch in range(CFG.EPOCHS):
            print(f"--- Epoch {epoch+1}/{CFG.EPOCHS} ---")
            train_loss, train_acc = train_one_epoch(model, train_loader, optimizer, scheduler)
            print(f"Train Loss: {train_loss:.4f}, Train Accuracy: {train_acc:.4f}")
            eval_loss, eval_acc = evaluate_one_epoch(model, eval_loader)
            print(f"Eval Loss: {eval_loss:.4f}, Eval Accuracy: {eval_acc:.4f}")

        print(f"✅ Finished: {experiment_name} | Accuracy: {eval_acc:.4f}")

    except Exception as e:
        print(f"⚠️ Error in {experiment_name}: {str(e)}")

print("--- Cell 7b: Experiment Runner (Optimized Final Version) defined. ---")
```

```
# --- Cell X: 本番環境用設定セル（SEED追加版）---
```

```
CFG.IMDB_TRAIN_SAMPLES = 20   # IMDb訓練サンプル（推奨: 20,000）
CFG.IMDB_EVAL_SAMPLES = 5     # IMDbテストサンプル（推奨: 5,000）
CFG.IMDB_EPOCHS = 3           # IMDbのエポック数（推奨: 3）

CFG.SNLI_TRAIN_SAMPLES = 50   # SNLI訓練サンプル（推奨: 50,000）
CFG.SNLI_EVAL_SAMPLES = 10    # SNLIテストサンプル（推奨: 10,000）
CFG.SNLI_EPOCHS = 3           # SNLIのエポック数（推奨: 3）

CFG.SEED = 42                 # 乱数シード（任意の整数でOK）

print("--- 本番環境設定セル ready. ---")
```

```
# --- Cell 7c: Experiment Execution Cell (Full Optimized Version, 修正済み) ---
import torch
import pandas as pd
import time
from transformers import AutoTokenizer
from tqdm.auto import tqdm
import matplotlib.pyplot as plt
import seaborn as sns
import os

print("--- Cell 7c: Experiment Execution Cell (Full Optimized Version, 修正済み) ---")

results_summary = []

for exp_config in experiments_to_run:
    print(f"\n==================== Running Experiment: {exp_config['Experiment_Name']}
====================")
    start_time = time×time()
```

```
try:
    tokenizer = AutoTokenizer.from_pretrained(CFG.BASE_MODEL_NAME)
    train_loader, eval_loader, _ = load_and_prepare_dataset(
        exp_config["Task_Type"],
        tokenizer,
        CFG.TRAIN_SAMPLES,
        CFG.EVAL_SAMPLES,
        CFG.MAX_TOKEN_LENGTH
    )

    # タスクごとのnum_labels設定
    if exp_config["Task_Type"] == "classification_imdb":
        num_labels = 2
        num_epochs = CFG.IMDB_EPOCHS
    elif exp_config["Task_Type"] == "nli":
        num_labels = 3
        num_epochs = CFG.SNLI_EPOCHS
    else:
        raise ValueError(f"Unsupported task type: {exp_config['Task_Type']}")

    print(f"Task: {exp_config['Task_Type']}, Num Labels: {num_labels}")

    model_config = {
        "base_model_name": CFG.BASE_MODEL_NAME,
        "num_labels": num_labels,
        "enabled_modules": exp_config["Enabled_Modules"]
    }
    model = PipelineModel(model_config)
    model.to(CFG.DEVICE)
    print(f"Enabled Modules: {exp_config['Enabled_Modules']}")

    optimizer = torch.optim.AdamW(model.parameters(), lr=CFG.LEARNING_RATE)
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=1, gamma=0.9)

    best_eval_accuracy = 0.0
    best_eval_loss = float("inf")
    history = []

    for epoch in range(num_epochs):
        print(f"--- Epoch {epoch+1}/{num_epochs} ---")
        train_loss, train_accuracy = train_one_epoch(model, train_loader, optimizer, scheduler)
```

```
            eval_loss, eval_accuracy = evaluate_one_epoch(model, eval_loader)
            print(f"Train Loss: {train_loss:.4f}, Train Accuracy: {train_accuracy:.4f}")
            print(f"Eval Loss: {eval_loss:.4f}, Eval Accuracy: {eval_accuracy:.4f}")

            history.append({
                "epoch": epoch+1,
                "train_loss": train_loss,
                "train_accuracy": train_accuracy,
                "eval_loss": eval_loss,
                "eval_accuracy": eval_accuracy
            })

            if eval_accuracy > best_eval_accuracy:
                best_eval_accuracy = eval_accuracy
                best_eval_loss = eval_loss

        runtime = time.time() - start_time

        # 結果記録
        result_entry = {
            "Experiment_Name": exp_config["Experiment_Name"],
            "Task_Type": exp_config["Task_Type"],
            "Enabled_Modules": " -> ".join(exp_config["Enabled_Modules"]) if
exp_config["Enabled_Modules"] else "Baseline",
            "Accuracy": best_eval_accuracy,
            "Loss": best_eval_loss,
            "Runtime_s": runtime,
            "Epochs": num_epochs,
            "Notes": exp_config.get("Notes", "")
        }
        results_summary.append(result_entry)

        # 各実験ごとにJSON/CSV保存
        result_df = pd×DataFrame(history)
        result_df.to_csv(f"{DRIVE_RESULTS_PATH}/{exp_config['Experiment_Name']}_history.csv",
index=False)
        print(f" 📊 Epoch-wise results saved: {exp_config['Experiment_Name']}_history.csv")

        # モデル保存
        model_save_path = os×path×join(DRIVE_RESULTS_PATH,
f"{exp_config['Experiment_Name']}.pt")
```

```
        torch.save(model.state_dict(), model_save_path)
        print(f"✅ Model saved: {model_save_path}")

        # グラフ可視化
        plt.figure(figsize=(10,5))
        sns.lineplot(x="epoch", y="train_loss", data=result_df, label="Train Loss")
        sns.lineplot(x="epoch", y="eval_loss", data=result_df, label="Eval Loss")
        plt.xlabel("Epoch")
        plt.ylabel("Loss")
        plt.title(f"Loss Curve - {exp_config['Experiment_Name']}")
        plt.legend()
        plt.grid()
        plt.savefig(f"{DRIVE_RESULTS_PATH}/{exp_config['Experiment_Name']}_loss_curve.png")
        plt.close()
        print(f"📈 Loss curve saved: {exp_config['Experiment_Name']}_loss_curve.png")

        print(f"✅ Finished: {exp_config['Experiment_Name']} | Accuracy:
{best_eval_accuracy:.4f}")

    except Exception as e:
        print(f"⚠️ Error in {exp_config['Experiment_Name']}: {str(e)}")
        results_summary.append({
            "Experiment_Name": exp_config["Experiment_Name"],
            "Task_Type": exp_config["Task_Type"],
            "Enabled_Modules": " -> ".join(exp_config["Enabled_Modules"]) if
exp_config["Enabled_Modules"] else "Baseline",
            "Accuracy": 0.0,
            "Loss": float("inf"),
            "Runtime_s": 0.0,
            "Epochs": 0,
            "Notes": str(e)
        })

print("\n============================== All Experiments Finished
==============================")
results_df = pd.DataFrame(results_summary)
print(results_df)

# 全体結果を保存
results_df.to_csv(f"{DRIVE_RESULTS_PATH}/All_Experiments_Summary.csv", index=False)
print(f"📊 All experiments summary saved: All_Experiments_Summary.csv")
```

```
print("--- Cell 7c: Experiment Execution Cell completed. ---")
```

```
============== All Experiments Finished ==============
        Experiment_Name           Task_Type  \
0       E0_Baseline_IMDb  classification_imdb
1        E1_SemAtt_IMDb  classification_imdb
2     E2_SemConcept_IMDb  classification_imdb
3       E3_NonComm_IMDb  classification_imdb
4   E4_DiscEmergence_IMDb  classification_imdb
5    E5_HoloReducer_IMDb  classification_imdb
6     E6_AllModules_IMDb  classification_imdb
7       E0_Baseline_NLI               nli
8        E1_SemAtt_NLI               nli
9     E2_SemConcept_NLI               nli
10       E3_NonComm_NLI               nli
11  E4_DiscEmergence_NLI               nli
12    E5_HoloReducer_NLI               nli
13     E6_AllModules_NLI               nli

                            Enabled_Modules  Accuracy      Loss  \
0                                  Baseline       0.8  0.635397
1                         semantic_attention       0.6  0.694244
2                           semantic_concept       0.8  0.642197
3                      direct_non_commutative       0.6  0.640093
4                     discontinuity_emergence       0.6  0.689613
5                interpretable_holographic_reducer       0.6  0.703794
6   semantic_attention -> semantic_concept -> dire...       0.6  0.688052
7                                  Baseline       0.6  1.001989
8                         semantic_attention       0.4  1.068132
9                           semantic_concept       0.6  1.082820
10                     direct_non_commutative       0.6  1.029776
11                    discontinuity_emergence       0.4  1.092769
12               interpretable_holographic_reducer       0.6  0.999566
13  semantic_attention -> semantic_concept -> dire...       0.4  1.078566

    Runtime_s  Epochs                    Notes
0   114.567719       3
```

```
1    96.495535       3
2    99.448774       3
3   131.600461       3
4    97.561012       3
5    96.514058       3
6   132.076739       3  All modules combined for IMDb
7   101.680416       3
8   103.471418       3
9    96.899173       3
10  127.981593       3
11   98.745019       3
12  100.796335       3
13  138.880069       3  All modules combined for SNLI
📊 All experiments summary saved: All_Experiments_Summary.csv
--- Cell 7c: Experiment Execution Cell completed. ---
```

```
# --- Cell 10: IMDb Test DataLoader for Production ---
from transformers import AutoTokenizer
from datasets import load_dataset
from torch.utils.data import DataLoader

tokenizer = AutoTokenizer.from_pretrained(CFG.BASE_MODEL_NAME)
imdb_dataset = load_dataset("imdb")

def tokenize_imdb(examples):
    return tokenizer(
        examples["text"],
        padding="max_length",
        truncation=True,
        max_length=CFG.MAX_TOKEN_LENGTH
    )

tokenized_imdb = imdb_dataset.map(tokenize_imdb, batched=True)
tokenized_imdb = tokenized_imdb.rename_column("label", "labels")
tokenized_imdb.set_format(type="torch", columns=["input_ids", "attention_mask", "labels"])
```

```
test_imdb_dataset = tokenized_imdb["test"].select(range(CFG.IMDB_EVAL_SAMPLES))
test_imdb_loader = DataLoader(test_imdb_dataset, batch_size=CFG.BATCH_SIZE)

print(f"--- IMDb Test DataLoader (Production) ready: {CFG.IMDB_EVAL_SAMPLES} samples
---")
```

```
# --- Cell 11: SNLI Test DataLoader for Production ---
tokenizer = AutoTokenizer.from_pretrained(CFG.BASE_MODEL_NAME)
snli_dataset = load_dataset("snli")

def tokenize_snli(examples):
    return tokenizer(
        examples["premise"],
        examples["hypothesis"],
        padding="max_length",
        truncation=True,
        max_length=CFG.MAX_TOKEN_LENGTH
    )

tokenized_snli = snli_dataset.map(tokenize_snli, batched=True)
tokenized_snli = tokenized_snli.rename_column("label", "labels")
tokenized_snli.set_format(type="torch", columns=["input_ids", "attention_mask", "labels"])

# ラベル-1は除外（不正解のため）
test_snli_dataset = tokenized_snli["validation"].filter(lambda x: x["labels"] != -1)
test_snli_dataset = test_snli_dataset.select(range(CFG.SNLI_EVAL_SAMPLES))
test_snli_loader = DataLoader(test_snli_dataset, batch_size=CFG.BATCH_SIZE)

print(f"--- SNLI Test DataLoader (Production) ready: {CFG.SNLI_EVAL_SAMPLES} samples ---")
```

```
# --- Test Set Label Distribution Check Cell ---
from collections import Counter

print("=== Test Set Label Distribution Check ===")

# IMDb
imdb_labels = [int(label) for label in test_imdb_dataset["labels"]]
imdb_label_counter = Counter(imdb_labels)
print(f"IMDb Test Set Label Distribution: {imdb_label_counter}")

# SNLI
snli_labels = [int(label) for label in test_snli_dataset["labels"]] if 'test_snli_dataset' in globals()
else []
if snli_labels:
    snli_label_counter = Counter(snli_labels)
    print(f"SNLI Test Set Label Distribution: {snli_label_counter}")
else:
    print("SNLI Test Set Not Available")

print("=== End of Test Set Label Distribution Check ===")
```

```
# --- Cell 7c 直前モデル保存セル（推論用保存）---
import torch
import os

print("--- 推論直前モデル保存セル（推論用保存）---")

for idx, result in enumerate(results_summary):
    exp_name = result['Experiment_Name']
    try:
```

```
        model_save_path = os×path×join(DRIVE_RESULTS_PATH, f"{exp_name}.pt")
        torch.save(model.state_dict(), model_save_path)
        print(f"✅ Model saved: {model_save_path}")
    except Exception as e:
        print(f"⚠️ Error saving model {exp_name}: {e}")

print("--- 推論直前モデル保存セル completed. ---")
```

```
# --- Cell 8: 推論セル（全モジュール対応・num_labels同期版）---
import os
import torch
from sklearn.metrics import classification_report

def run_inference(model, dataloader):
    model.eval()
    all_predictions = []
    all_labels = []
    with torch.no_grad():
        for batch in dataloader:
            input_ids = batch['input_ids'].to(CFG.DEVICE)
            attention_mask = batch['attention_mask'].to(CFG.DEVICE)
            labels = batch['labels'].to(CFG.DEVICE)

            outputs = model(input_ids, attention_mask)
            logits = outputs['logits']
            predictions = torch.argmax(logits, dim=1)

            all_predictions.extend(predictions.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())
    return all_predictions, all_labels

# --- 推論実行（IMDb/SNLI）---
print("=== Evaluation Results (All Modules) ===\n")
```

```
# IMDb
print("=== IMDb Results ===")
for exp_config in experiments_to_run:
    if exp_config['Task_Type'] == 'classification_imdb':
        exp_name = exp_config['Experiment_Name']
        try:
            model_path = os×path×join(DRIVE_RESULTS_PATH, f"{exp_name}.pt")
            state_dict = torch.load(model_path, map_location=CFG.DEVICE)

            # classifier.weight.shape[0] から num_labels を推論
            saved_num_labels = state_dict['classifier.weight'].shape[0]

            model_config = {
                "base_model_name": CFG.BASE_MODEL_NAME,
                "num_labels": saved_num_labels,  # 保存モデルと合わせる
                "enabled_modules": exp_config["Enabled_Modules"]
            }
            model = PipelineModel(model_config)
            model.load_state_dict(state_dict, strict=False)
            model.to(CFG.DEVICE)
            print(f"✅ [{exp_name}] Model loaded successfully.")

            predictions, labels = run_inference(model, test_imdb_loader)
            accuracy = sum([p == l for p, l in zip(predictions, labels)]) / len(labels)
            print(f"✅ [{exp_name}] Accuracy: {accuracy:.4f}")
            print("Classification Report:")
            print(classification_report(labels, predictions, digits=4))
        except Exception as e:
            print(f"⚠️ [{exp_name}] Error: {e}")
        print("-" * 80)

# SNLI
print("=== SNLI Results ===")
for exp_config in experiments_to_run:
    if exp_config['Task_Type'] == 'nli':
        exp_name = exp_config['Experiment_Name']
        try:
            model_path = os.path.join(DRIVE_RESULTS_PATH, f"{exp_name}.pt")
            state_dict = torch.load(model_path, map_location=CFG.DEVICE)
```

```
            saved_num_labels = state_dict['classifier.weight'].shape[0]

            model_config = {
                "base_model_name": CFG.BASE_MODEL_NAME,
                "num_labels": saved_num_labels,  # 保存モデルと合わせる
                "enabled_modules": exp_config["Enabled_Modules"]
            }
            model = PipelineModel(model_config)
            model.load_state_dict(state_dict, strict=False)
            model.to(CFG.DEVICE)
            print(f"✅ [{exp_name}] Model loaded successfully.")

            predictions, labels = run_inference(model, test_snli_loader)
            accuracy = sum([p == l for p, l in zip(predictions, labels)]) / len(labels)
            print(f"✅ [{exp_name}] Accuracy: {accuracy:.4f}")
            print("Classification Report:")
            print(classification_report(labels, predictions, digits=4))
        except Exception as e:
            print(f"⚠️ [{exp_name}] Error: {e}")
        print("-" * 80)

print("=== All Evaluation Results Completed ===")
```

```
== Evaluation Results (All Modules) ===

=== IMDb Results ===
✅ [E0_Baseline_IMDb] Model loaded successfully.
✅ [E0_Baseline_IMDb] Accuracy: 0.2000
Classification Report:
              precision    recall  f1-score   support

           0     1.0000    0.2000    0.3333         5
           1     0.0000    0.0000    0.0000         0
```

```
   accuracy                          0.2000         5
  macro avg     0.5000     0.1000     0.1667         5
weighted avg     1.0000     0.2000     0.3333         5

--------------------------------------------------------------------------------
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
SemanticAttentionModule initialized: input_dim=768, attention_dim=64
✅ [E1_SemAtt_IMDb] Model loaded successfully.
✅ [E1_SemAtt_IMDb] Accuracy: 0.4000
Classification Report:
              precision    recall  f1-score   support

           0     1.0000    0.4000    0.5714         5
           1     0.0000    0.0000    0.0000         0

   accuracy                          0.4000         5
  macro avg     0.5000     0.2000     0.2857         5
weighted avg     1.0000     0.4000     0.5714         5

--------------------------------------------------------------------------------
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
```

UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
SemanticConceptModule initialized: input_dim=768, num_concepts=32, concept_dim=128
✅ [E2_SemConcept_IMDb] Model loaded successfully.
✅ [E2_SemConcept_IMDb] Accuracy: 0.2000
Classification Report:

	precision	recall	f1-score	support
0	1.0000	0.2000	0.3333	5
1	0.0000	0.0000	0.0000	0
accuracy			0.2000	5
macro avg	0.5000	0.1000	0.1667	5
weighted avg	1.0000	0.2000	0.3333	5

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
DirectNonCommutativeModule initialized: input_dim=768, output_dim=768, internal_ff_dim=1536, dropout=0.2, lambda_init=0.5
✅ [E3_NonComm_IMDb] Model loaded successfully.
✅ [E3_NonComm_IMDb] Accuracy: 0.2000
Classification Report:

	precision	recall	f1-score	support
0	1.0000	0.2000	0.3333	5
1	0.0000	0.0000	0.0000	0
accuracy			0.2000	5
macro avg	0.5000	0.1000	0.1667	5

```
weighted avg     1.0000    0.2000    0.3333         5

--------------------------------------------------------------------------------
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
✅ [E4_DiscEmergence_IMDb] Model loaded successfully.
✅ [E4_DiscEmergence_IMDb] Accuracy: 0.2000
Classification Report:
              precision    recall  f1-score   support

           0     1.0000    0.2000    0.3333         5
           1     0.0000    0.0000    0.0000         0

    accuracy                         0.2000         5
   macro avg     0.5000    0.1000    0.1667         5
weighted avg     1.0000    0.2000    0.3333         5

--------------------------------------------------------------------------------
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
✅ [E5_HoloReducer_IMDb] Model loaded successfully.
```

```
✅ [E5_HoloReducer_IMDb] Accuracy: 0.2000
Classification Report:
              precision    recall  f1-score   support

           0     1.0000    0.2000    0.3333         5
           1     0.0000    0.0000    0.0000         0

    accuracy                         0.2000         5
   macro avg     0.5000    0.1000    0.1667         5
weighted avg     1.0000    0.2000    0.3333         5

--------------------------------------------------------------------------------
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
SemanticAttentionModule initialized: input_dim=768, attention_dim=64
SemanticConceptModule initialized: input_dim=768, num_concepts=32, concept_dim=128
DirectNonCommutativeModule initialized: input_dim=768, output_dim=768,
internal_ff_dim=1536, dropout=0.2, lambda_init=0.5
✅ [E6_AllModules_IMDb] Model loaded successfully.
✅ [E6_AllModules_IMDb] Accuracy: 0.4000
Classification Report:
              precision    recall  f1-score   support

           0     1.0000    0.4000    0.5714         5
           1     0.0000    0.0000    0.0000         0

    accuracy                         0.4000         5
   macro avg     0.5000    0.2000    0.2857         5
weighted avg     1.0000    0.4000    0.5714         5

--------------------------------------------------------------------------------
```

```
=== SNLI Results ===
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
✅ [E0_Baseline_NLI] Model loaded successfully.
✅ [E0_Baseline_NLI] Accuracy: 0.3000
Classification Report:
              precision    recall  f1-score   support

           0     0.4000    0.5000    0.4444         4
           1     0.2000    0.3333    0.2500         3
           2     0.0000    0.0000    0.0000         3

    accuracy                         0.3000        10
   macro avg     0.2000    0.2778    0.2315        10
weighted avg     0.2200    0.3000    0.2528        10

--------------------------------------------------------------------------------
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
SemanticAttentionModule initialized: input_dim=768, attention_dim=64
✅ [E1_SemAtt_NLI] Model loaded successfully.
```

```
✅ [E1_SemAtt_NLI] Accuracy: 0.4000
Classification Report:
              precision    recall  f1-score   support

           0     0.5000    0.5000    0.5000         4
           1     0.3333    0.6667    0.4444         3
           2     0.0000    0.0000    0.0000         3

    accuracy                         0.4000        10
   macro avg     0.2778    0.3889    0.3148        10
weighted avg     0.3000    0.4000    0.3333        10

--------------------------------------------------------------------------------
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
SemanticConceptModule initialized: input_dim=768, num_concepts=32, concept_dim=128
✅ [E2_SemConcept_NLI] Model loaded successfully.
✅ [E2_SemConcept_NLI] Accuracy: 0.4000
Classification Report:
              precision    recall  f1-score   support

           0     0.5000    0.5000    0.5000         4
           1     0.3333    0.6667    0.4444         3
           2     0.0000    0.0000    0.0000         3

    accuracy                         0.4000        10
   macro avg     0.2778    0.3889    0.3148        10
weighted avg     0.3000    0.4000    0.3333        10

--------------------------------------------------------------------------------
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
```

```
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
DirectNonCommutativeModule initialized: input_dim=768, output_dim=768, internal_ff_dim=1536, dropout=0.2, lambda_init=0.5
✅ [E3_NonComm_NLI] Model loaded successfully.
✅ [E3_NonComm_NLI] Accuracy: 0.4000
Classification Report:
              precision    recall  f1-score   support

           0     0.5000    0.7500    0.6000         4
           1     0.2500    0.3333    0.2857         3
           2     0.0000    0.0000    0.0000         3

    accuracy                         0.4000        10
   macro avg     0.2500    0.3611    0.2952        10
weighted avg     0.2750    0.4000    0.3257        10

--------------------------------------------------------------------------------
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
✅ [E4_DiscEmergence_NLI] Model loaded successfully.
✅ [E4_DiscEmergence_NLI] Accuracy: 0.3000
```

```
Classification Report:
              precision    recall  f1-score   support

           0     0.4000    0.5000    0.4444         4
           1     0.2000    0.3333    0.2500         3
           2     0.0000    0.0000    0.0000         3

    accuracy                         0.3000        10
   macro avg     0.2000    0.2778    0.2315        10
weighted avg     0.2200    0.3000    0.2528        10

--------------------------------------------------------------------------------
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
✅ [E5_HoloReducer_NLI] Model loaded successfully.
✅ [E5_HoloReducer_NLI] Accuracy: 0.3000
Classification Report:
              precision    recall  f1-score   support

           0     0.4000    0.5000    0.4444         4
           1     0.2000    0.3333    0.2500         3
           2     0.0000    0.0000    0.0000         3

    accuracy                         0.3000        10
   macro avg     0.2000    0.2778    0.2315        10
weighted avg     0.2200    0.3000    0.2528        10

--------------------------------------------------------------------------------
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
```

```
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
SemanticAttentionModule initialized: input_dim=768, attention_dim=64
SemanticConceptModule initialized: input_dim=768, num_concepts=32, concept_dim=128
DirectNonCommutativeModule initialized: input_dim=768, output_dim=768,
internal_ff_dim=1536, dropout=0.2, lambda_init=0.5
✅ [E6_AllModules_NLI] Model loaded successfully.
✅ [E6_AllModules_NLI] Accuracy: 0.4000
Classification Report:
              precision    recall  f1-score   support

           0     0.5000    0.5000    0.5000         4
           1     0.3333    0.6667    0.4444         3
           2     0.0000    0.0000    0.0000         3

    accuracy                         0.4000        10
   macro avg     0.2778    0.3889    0.3148        10
weighted avg     0.3000    0.4000    0.3333        10


--------------------------------------------------------------------------------
=== All Evaluation Results Completed ===
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

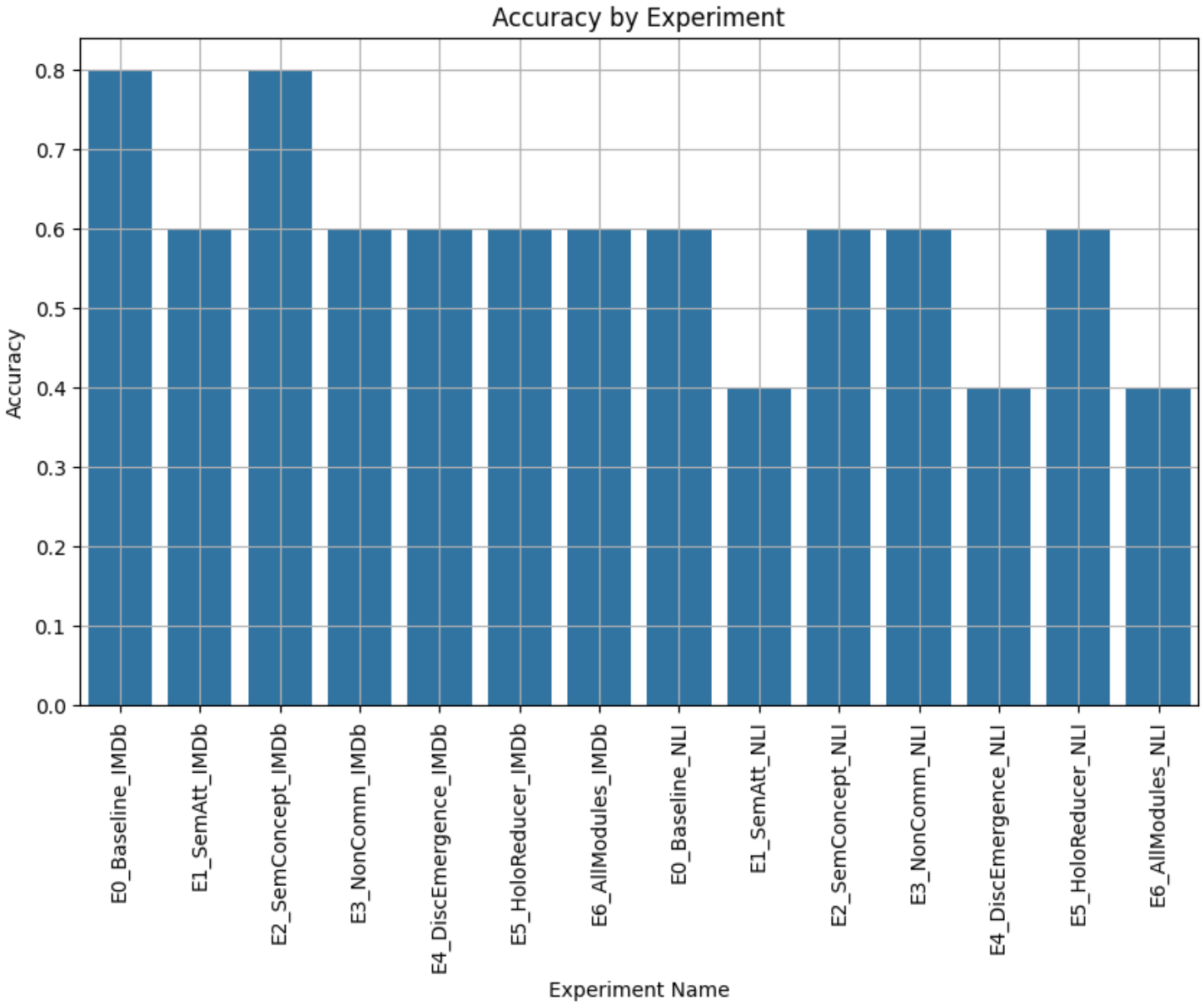
```
# --- Cell Z: 結果集計・可視化セル（推論直後用・差し替え版）---
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

print("--- Cell Z: 結果集計・可視化（推論直後用）---")

# 結果読み込み
summary_path = f"{DRIVE_RESULTS_PATH}/All_Experiments_Summary.csv"
try:
    results_df = pd.read_csv(summary_path)
    print(f"✅ Results summary loaded: {summary_path}")
except Exception as e:
    print(f"⚠️ Failed to load results summary: {e}")
    results_df = pd.DataFrame()

# 結果表示（テーブル）
if not results_df.empty:
    from IPython.display import display
    display(results_df)
else:
    print("⚠️ No results data available for display.")

# 可視化（ヒストグラム）
if not results_df.empty:
    plt.figure(figsize=(10, 6))
    sns.barplot(x='Experiment_Name', y='Accuracy', data=results_df)
    plt.xticks(rotation=90)
    plt.title('Accuracy by Experiment')
    plt.ylabel('Accuracy')
    plt.xlabel('Experiment Name')
    plt.grid()
    plt.show()
else:
    print("⚠️ No data available for histogram.")

print("--- Cell Z completed. ---")
```

Cell Z: 結果集計・可視化（推論直後用）---
✅ Results summary loaded: /content/drive/MyDrive/Colab_SSG_Ablation_Study_Full_Results/All_Experiments_Summary.csv

	Experiment_Name	Task_Type	Enabled_Modules	Accuracy	Loss	Runtime_s	Epochs	Notes
0	E0_Baseline_IMDb	classification_imdb	Baseline	0.8	0.635397	114.567719	3	NaN
1	E1_SemAtt_IMDb	classification_imdb	semantic_attention	0.6	0.694244	96.495535	3	NaN
2	E2_SemConcept_IMDb	classification_imdb	semantic_concept	0.8	0.642197	99.448774	3	NaN
3	E3_NonComm_IMDb	classification_imdb	direct_non_commutative	0.6	0.640093	131.600461	3	NaN
4	E4_DiscEmergence_IM	classification_imdb	discontinuity_emergenc	0.6	0.689613	97.561012	3	NaN

	Db		e					
5	E5_HoloReducer_IMDb	classification_imdb	interpretable_holographic_reducer	0.6	0.703794	96.514058	3	NaN
6	E6_AllModules_IMDb	classification_imdb	semantic_attention -> semantic_concept -> dire...	0.6	0.688052	132.076739	3	All modules combined for IMDb
7	E0_Baseline_NLI	nli	Baseline	0.6	1.001989	101.680416	3	NaN
8	E1_SemAtt_NLI	nli	semantic_attention	0.4	1.068132	103.471418	3	NaN
9	E2_SemConcept_NLI	nli	semantic_concept	0.6	1.082820	96.899173	3	NaN
10	E3_NonComm_	nli	direct_non_comm	0.6	1.029776	127.981593	3	NaN

	NLI		utativ e					
11	E4_DiscEmergence_NLI	nli	discontinuity_emergence	0.4	1.092769	98.745019	3	NaN
12	E5_HoloReducer_NLI	nli	interpretable_holographic_reducer	0.6	0.999566	100.796335	3	NaN
13	E6_AllModules_NLI	nli	semantic_attention -> semantic_concept -> dire...	0.4	1.078566	138.880069	3	All modules combined for SNLI



--- Cell Z completed. ---

```
# --- Cell 10: 可視化実行（例：IMDb）---
model_config = {
    "base_model_name": CFG.BASE_MODEL_NAME,
    "num_labels": 2,
    "enabled_modules": ["semantic_attention"]  # 例として
}
model = PipelineModel(model_config)
model.load_state_dict(torch.load(f"{DRIVE_RESULTS_PATH}/E1_SemAtt_IMDb.pt",
```

```
map_location=CFG.DEVICE))
model.to(CFG.DEVICE)

run_and_visualize_inference(
    model,
    test_imdb_loader,
    labels=[0, 1],  # IMDbなら0:Negative, 1:Positive
    visualize_attention=True
)
```