査読論文 本文構成案：「トランスフォーマーにおける解釈可能な意味概念と非可換性処理の導入効果」

統合アブレーション実験コード（ホログラフ縮減）

エコツーラボ合同会社
猪澤也寸志
polyp@ webman.jp
（2025年5月28日現地時間JST）

セル0

```
# セル1 (セットアップとカーネル再起動用)
# ----------------------------------
# 関連ライブラリを一度アンインストールしてクリーンな状態にする
!pip uninstall -y transformers accelerate peft sentence-transformers datasets -q

# 指定バージョンでライブラリをインストール
!pip install transformers==4.36.2 -q
!pip install datasets -q
!pip install peft==0.7.1 -q
!pip install accelerate==0.25.0 -q  # <--- accelerateのバージョンを0.25.0に固定 (または0.26.1)

# 変更を適用するためにカーネルを再起動
import os
os.kill(os.getpid(), 9)
```

セル1

```
# キャッシュクリア用コマンドの例
!rm -rf ~/.cache/huggingface/datasets/imdb
print("IMDb dataset cache cleared.")
```

セル2

```
# --- セル2: ライブラリのインポート ---

import torch
import torch.nn as nn
import torch.nn.functional as F # SemanticConceptModule で F.softmax を使用するため

from transformers import (
    AutoModelForSequenceClassification,
    AutoTokenizer,
    Trainer,
    TrainingArguments,
    AutoConfig,
    PreTrainedModel,
    AutoModel
)
from transformers.modeling_outputs import SequenceClassifierOutput # モデルの出力形式用
```

```
from torch.nn import CrossEntropyLoss # 損失関数用

from datasets import load_dataset, Dataset # ★ Dataset クラスをインポート

import numpy as np
from sklearn.metrics import accuracy_score # 評価指標計算用
import itertools # 実験構成生成用 (今回は手動定義なので厳密には不要ですが、将来的に使う可能性も)

# (もし他に必要な標準ライブラリがあればここに追加してください)
# import os # (セル1で使用済みなので、ここでは必須ではない)
# import json
# import time
# import datetime

# (もしグラフ描画などを行う場合は、matplotlibなどもここでインポート)
# import matplotlib.pyplot as plt
# import seaborn as sns
# import pandas as pd

print("Cell 2: Libraries imported successfully.")
print(f"  PyTorch version: {torch.__version__}")
try:
    import transformers
    print(f"  Transformers version: {transformers.__version__}")
    import datasets
    print(f"  Datasets version: {datasets.__version__}")
    import accelerate
    print(f"  Accelerate version: {accelerate.__version__}")
    import peft
    print(f"  PEFT version: {peft.__version__}")
except ImportError:
    print("Warning: Could not print all Hugging Face library versions. Ensure they are installed.")


セル 3

# --- セル3: 基本設定・共通パラメータ ---

from transformers import AutoTokenizer, AutoConfig # 必要なものをインポート (セル2でインポート済みなら重複を避けてもOK)

# --- 基本的な設定値 ---
BASE_MODEL_NAME = "bert-base-uncased"  # ベースとなる事前学習済みモデルの名前
NUM_LABELS = 2                  # 分類タスクのラベル数 (例: IMDbならポジティブ/ネガティブの2)
OUTPUT_DIR_BASE = "./experiment_results/" # 実験結果の出力先ベースディレクトリ

# --- 訓練に関するハイパーパラメータ ---
LEARNING_RATE = 2e-5            # 学習率
BATCH_SIZE = 1                  # バッチサイズ (GPUメモリやデータに応じて調整)
                                # CPU実行の場合は小さめ(例: 1や2)にしないと非常に遅い可能性があります
NUM_EPOCHS = 1                  # 訓練エポック数 (最初は1などでテストし、問題なければ増やす)

# --- トークナイザーとモデル設定のロード ---
try:
    print(f"Loading tokenizer for {BASE_MODEL_NAME}...")
    tokenizer = AutoTokenizer.from_pretrained(BASE_MODEL_NAME)
    print(f"Tokenizer for {BASE_MODEL_NAME} loaded successfully.")
except Exception as e:
    print(f"Error loading tokenizer for {BASE_MODEL_NAME}: {e}")
    raise # エラーが発生したら処理を停止

try:
```

```
    print(f"Loading Hugging Face model config for {BASE_MODEL_NAME}...")
    hf_model_config = AutoConfig.from_pretrained(
        BASE_MODEL_NAME,
        num_labels=NUM_LABELS
        # 必要に応じて他のAutoConfigのパラメータもここで設定可能
        # 例: finetuning_task='text-classification' など
    )
    print(f"Model config for {BASE_MODEL_NAME} loaded successfully.")
except Exception as e:
    print(f"Error loading model config for {BASE_MODEL_NAME}: {e}")
    raise

print("\n--- Cell 3: Basic/Common Settings Defined ---")
print(f"BASE_MODEL_NAME: {BASE_MODEL_NAME}")
print(f"NUM_LABELS: {NUM_LABELS}")
print(f"OUTPUT_DIR_BASE: {OUTPUT_DIR_BASE}")
print(f"LEARNING_RATE: {LEARNING_RATE}")
print(f"BATCH_SIZE: {BATCH_SIZE}")
print(f"NUM_EPOCHS: {NUM_EPOCHS}")
print(f"Tokenizer type: {type(tokenizer).__name__}")
print(f"HF Model Config type: {type(hf_model_config).__name__}")
```

セル 4

```
# --- セル4: データセット準備 と compute_metrics 関数の定義 (本格運用版) ---

from datasets import load_dataset
from sklearn.metrics import accuracy_score
import numpy as np
import torch

# --- グローバル変数 (セル3で定義されているはずのもの) の確認 ---
if 'BASE_MODEL_NAME' not in globals():
    raise NameError("Global variable 'BASE_MODEL_NAME' is not defined. Please ensure Cell 3 has been executed.")
if 'tokenizer' not in globals():
    raise NameError("Global variable 'tokenizer' is not defined. Please ensure Cell 3 has been executed.")

print(f"Starting dataset preparation using BASE_MODEL_NAME: {BASE_MODEL_NAME} and tokenizer:
{type(tokenizer).__name__}")

# 1. データセットのロード
try:
    dataset_dict = load_dataset("imdb") # DatasetDictオブジェクトとしてロード
    print("IMDb dataset loaded successfully.")
except Exception as e:
    print(f"Error loading IMDb dataset: {e}")
    raise

# 2. トークナイズ関数の定義
def tokenize_function(examples):
    return tokenizer(examples["text"], padding="max_length", truncation=True, max_length=256)

# 3. データセットのトークナイズ
try:
    print("Tokenizing datasets...")
    tokenized_train = dataset_dict["train"].map(tokenize_function, batched=True, remove_columns=["text"])
    tokenized_test = dataset_dict["test"].map(tokenize_function, batched=True, remove_columns=["text"])
    print("Tokenization complete for train and test splits.")
except Exception as e:
    print(f"Error during tokenization: {e}")
```

```
    raise

# 4. 訓練用・評価用データセットの準備とカラム名修正
try:
    # 'label' カラムを 'labels' にリネーム
    if 'label' in tokenized_train.column_names and 'labels' not in tokenized_train.column_names:
        print("Renaming 'label' to 'labels' in tokenized_train...")
        final_tokenized_train = tokenized_train.rename_column("label", "labels")
    else:
        final_tokenized_train = tokenized_train
        if 'labels' not in final_tokenized_train.column_names:
            print("Warning: 'labels' column expected but not found in final_tokenized_train.")


    if 'label' in tokenized_test.column_names and 'labels' not in tokenized_test.column_names:
        print("Renaming 'label' to 'labels' in tokenized_test...")
        final_tokenized_test = tokenized_test.rename_column("label", "labels")
    else:
        final_tokenized_test = tokenized_test
        if 'labels' not in final_tokenized_test.column_names:
            print("Warning: 'labels' column expected but not found in final_tokenized_test.")

    # --- ★★★ 本格的な実験用のデータサンプル数をここで設定 ★★★ ---
    NUM_TRAIN_SAMPLES_FULL = 10  # 例: 10,000件 (または len(final_tokenized_train) で全件)
    NUM_EVAL_SAMPLES_FULL = 2    # 例:  2,500件 (または len(final_tokenized_test) で全件)
    # ----------------------------------------------------------

    # グローバル変数として train_dataset と eval_dataset を定義
    train_dataset = final_tokenized_train.shuffle(seed=42).select(
        range(min(NUM_TRAIN_SAMPLES_FULL, len(final_tokenized_train)))
    )
    eval_dataset = final_tokenized_test.shuffle(seed=42).select(
        range(min(NUM_EVAL_SAMPLES_FULL, len(final_tokenized_test)))
    )

    print(f"Train dataset created. Number of samples: {len(train_dataset)}")
    print(f"  Train dataset FINAL column names: {train_dataset.column_names}")
    print(f"Evaluation dataset created. Number of samples: {len(eval_dataset)}")
    print(f"  Eval dataset FINAL column names: {eval_dataset.column_names}")

    if len(eval_dataset) > 0: # 念のため表示
        print(f"First sample of EVAL_dataset (for column check): {eval_dataset[0]}")

except Exception as e:
    print(f"Error creating or renaming train/eval datasets: {e}")
    raise

# 5. 評価指標計算関数の定義 (デバッグプリントは有効のまま残しておいても良い)
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    if labels is None or len(labels) == 0:
        print("--- Inside compute_metrics: Received no labels or labels is None. Returning empty metrics. ---")
        return {}
    predictions = np.argmax(logits, axis=-1)
    accuracy = accuracy_score(labels, predictions)
    metrics_dict = {"eval_accuracy": accuracy}
    # --- デバッグ用プリント (本格運用時はコメントアウトしても良い) ---
    # print(f"--- Inside compute_metrics ---")
    # print(f"  eval_pred logits type: {type(logits)}, labels type: {type(labels)}")
    # if hasattr(logits, 'shape'): print(f"  eval_pred logits shape: {logits.shape}")
    # if hasattr(labels, 'shape'): print(f"  eval_pred labels shape: {labels.shape}")
```

```
    # print(f"  Predictions example (first 5): {predictions[:5]}")
    # print(f"  Labels example (first 5): {labels[:5]}")
    # print(f"  Calculated accuracy: {accuracy}")
    # print(f"  Returning metrics_dict: {metrics_dict}")
    # --- ここまでデバッグ用プリント ---
    return metrics_dict

print("\nDataset preparation cell (Cell 4) complete.")
print(f"Defined global variables: 'train_dataset' (size: {len(train_dataset)}), 'eval_dataset' (size: {len(eval_dataset)}),
'compute_metrics'")
if callable(globals().get('compute_metrics')):
    print("'compute_metrics' function is defined and callable.")
else:
    print("Warning: 'compute_metrics' function is NOT defined or not callable.")
```

セル 5

```
# --- セル5: 意味アテンション機構 (SemanticConceptModule) の定義 ---
import torch
import torch.nn as nn
import torch.nn.functional as F
# from transformers import PretrainedConfig # model_configの型ヒント用 (必要なら)

class SemanticConceptModule(nn.Module):
    def __init__(self, input_dim, output_dim,
                 num_concepts=32, concept_dim=128,
                 model_config=None,
                 module_specific_config=None
                 ):
        super().__init__()
        self.input_dim = input_dim
        self.output_dim = output_dim
        self.num_concepts = num_concepts
        self.concept_dim = concept_dim

        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()

        if self.input_dim != self.output_dim:
            self.final_projection = nn.Linear(self.input_dim, self.output_dim)

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

    def forward(self, hidden_states, attention_mask=None, **kwargs):
        projected_hidden = self.activation(self.hidden_to_concept_proj(hidden_states))
        attention_scores = torch.matmul(projected_hidden, self.concept_prototypes.t())

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

        attention_weights = F.softmax(attention_scores, dim=-1)
        contextual_concepts = torch.matmul(attention_weights, self.concept_prototypes)
        projected_contextual_concepts = self.activation(self.concepts_to_integrate_proj(contextual_concepts))
        fused_hidden_states = hidden_states + projected_contextual_concepts
```

```
        if self.input_dim != self.output_dim:
            output = self.final_projection(fused_hidden_states)
        else:
            output = fused_hidden_states

        return {"last_hidden_state": output, "attention_weights": attention_weights}

    def get_output_dim(self):
        return self.output_dim

print("Cell 5: SemanticConceptModule class defined.")
```

セル 6

```
# --- セル6: 直接的非可換処理モジュール (DirectNonCommutativeModule) の定義 ---
import torch
import torch.nn as nn
# from transformers import PretrainedConfig # model_configの型ヒント用 (必要なら)

class DirectNonCommutativeModule(nn.Module):
    def __init__(self, input_dim, output_dim,
                 dropout=0.2,
                 model_config=None,
                 module_specific_config=None
                 ):
        super().__init__()
        self.input_dim = input_dim
        self.output_dim = output_dim
        internal_dropout = module_specific_config.get("dropout", dropout)
        internal_processing_dim = module_specific_config.get("internal_processing_dim", input_dim * 2)

        self.non_commutative_processor = nn.Sequential(
            nn.Linear(input_dim * 2, internal_processing_dim),
            nn.LayerNorm(internal_processing_dim),
            nn.ReLU(),
            nn.Dropout(internal_dropout),
            nn.Linear(internal_processing_dim, input_dim)
        )
        self.lambda_param = nn.Parameter(torch.tensor(0.5))

        if self.input_dim != self.output_dim:
            self.final_integration_projection = nn.Linear(self.input_dim, self.output_dim)

        print(f"DirectNonCommutativeModule defined and initialized: input_dim={input_dim}, output_dim={output_dim}")

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

        if seq_len < 2:
            # print("Warning: Seq_len < 2 in DirectNonCommutativeModule...") # 必要なら表示
            if self.input_dim == self.output_dim:
                return {"last_hidden_state": hidden_states, "non_commutative_effects_map": None}
            else:
                if not hasattr(self, 'final_integration_projection'):
                    temp_proj = nn.Linear(self.input_dim, self.output_dim).to(device)
                    return {"last_hidden_state": temp_proj(hidden_states), "non_commutative_effects_map": None}
                return {"last_hidden_state": self.final_integration_projection(hidden_states), "non_commutative_effects_map": None}
```

```
        non_commutative_effects_at_each_step = torch.zeros_like(hidden_states)
        for i in range(seq_len - 1):
            current_h = hidden_states[:, i, :]
            next_h = hidden_states[:, i + 1, :]
            forward_concat = torch.cat([current_h, next_h], dim=-1)
            backward_concat = torch.cat([next_h, current_h], dim=-1)
            forward_processed = self.non_commutative_processor(forward_concat)
            backward_processed = self.non_commutative_processor(backward_concat)
            non_comm_effect = self.lambda_param * (forward_processed - backward_processed)
            non_commutative_effects_at_each_step[:, i, :] += non_comm_effect

        fused_hidden_states = hidden_states + non_commutative_effects_at_each_step

        if self.input_dim != self.output_dim:
            output = self.final_integration_projection(fused_hidden_states)
        else:
            output = fused_hidden_states

        return {"last_hidden_state": output, "non_commutative_effects_map": non_commutative_effects_at_each_step}

    def get_output_dim(self):
        return self.output_dim

print("Cell 6: DirectNonCommutativeModule class defined.")
```

セル 7

```
# --- セル7: 断絶創発モジュール (DiscontinuityEmergenceModule) の定義 ---
import torch
import torch.nn as nn
# from transformers import PretrainedConfig # model_configの型ヒント用 (必要なら)

class DiscontinuityEmergenceModule(nn.Module):
    def __init__(self, input_dim, output_dim,
                 semantic_dim_for_discontinuity=12,
                 emergent_label_dim=24,
                 dropout=0.2,
                 model_config=None,
                 module_specific_config=None
                 ):
        super().__init__()
        self.input_dim = input_dim
        self.output_dim = output_dim
        self.semantic_dim_for_discontinuity = module_specific_config.get("semantic_dim_for_discontinuity",
semantic_dim_for_discontinuity)
        self.emergent_label_dim = module_specific_config.get("emergent_label_dim", emergent_label_dim)
        internal_dropout = module_specific_config.get("dropout", dropout)

        self.feature_to_semantic_for_discontinuity = nn.Sequential(
            nn.Linear(input_dim, input_dim // 2),
            nn.ReLU(),
            nn.Linear(input_dim // 2, self.semantic_dim_for_discontinuity),
            nn.Tanh()
        )
        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()
```

```
        )
        self.emergent_label_generator = nn.Sequential(
            nn.Linear(self.semantic_dim_for_discontinuity * 2, input_dim // 4),
            nn.LayerNorm(input_dim // 4),
            nn.ReLU(),
            nn.Dropout(internal_dropout),
            nn.Linear(input_dim // 4, self.emergent_label_dim),
            nn.Tanh()
        )
        if self.input_dim + self.emergent_label_dim != self.output_dim :
             self.integration_projection = nn.Linear(self.input_dim + self.emergent_label_dim, self.output_dim)
        elif self.input_dim != self.output_dim:
             self.integration_projection = nn.Linear(self.input_dim, self.output_dim)

        print(f"DiscontinuityEmergenceModule defined and initialized: input_dim={input_dim}, output_dim={output_dim}, "
              f"semantic_dim_disc={self.semantic_dim_for_discontinuity}, emergent_dim={self.emergent_label_dim}")

    def forward(self, hidden_states, attention_mask=None, **kwargs): # hidden_states を input_features から変更
        batch_size, seq_len, _ = hidden_states.shape
        device = hidden_states.device

        semantic_features_for_discontinuity = self.feature_to_semantic_for_discontinuity(hidden_states)

        discontinuity_scores = torch.zeros(batch_size, 0, device=device) # 初期化
        if seq_len > 1:
            sfd_t = semantic_features_for_discontinuity[:, :-1, :]
            sfd_t_plus_1 = semantic_features_for_discontinuity[:, 1:, :]
            combined_for_discontinuity = torch.cat([sfd_t, sfd_t_plus_1], dim=-1)
            discontinuity_scores = self.discontinuity_detector(combined_for_discontinuity)
            discontinuity_scores = discontinuity_scores.squeeze(-1)

        emergent_labels_sequence = torch.zeros(batch_size, 0, self.emergent_label_dim, device=device)
        if seq_len > 1:
            # combined_for_discontinuity を再利用
            emergent_labels_sequence = self.emergent_label_generator(combined_for_discontinuity)

        output_features = hidden_states
        if emergent_labels_sequence.numel() > 0 and emergent_labels_sequence.shape[1] > 0:
            pooled_emergent_label = emergent_labels_sequence.mean(dim=1)
            expanded_emergent_label = pooled_emergent_label.unsqueeze(1).expand(-1, seq_len, -1)
            combined_features = torch.cat([hidden_states, expanded_emergent_label], dim=-1)

            if hasattr(self, 'integration_projection'):
                output_features = self.integration_projection(combined_features)
            elif self.input_dim + self.emergent_label_dim == self.output_dim:
                 output_features = combined_features
            # ... (他のフォールバックや次元調整ロジックは以前のコードを参照)
        elif self.input_dim != self.output_dim :
            if not hasattr(self, 'final_projection_fallback'): # __init__で定義しておくべき
                self.final_projection_fallback = nn.Linear(self.input_dim, self.output_dim).to(device)
            output_features = self.final_projection_fallback(hidden_states)

        return {
            "last_hidden_state": output_features,
            "discontinuity_scores": discontinuity_scores,
            "emergent_labels_sequence": emergent_labels_sequence
        }

    def get_output_dim(self):
        return self.output_dim
```

```
print("Cell 7: DiscontinuityEmergenceModule class defined.")
```

セル８

```
# --- セル7.1: ホログラフ縮減モジュール (InterpretableHolographicReducer) の定義 ---
import torch
import torch.nn as nn
import torch.nn.functional as F
# from transformers import PretrainedConfig # model_configの型ヒント用 (必要なら)

class InterpretableHolographicReducer(nn.Module):
    def __init__(self, input_feature_dim, # 前の階層からの特徴ベクトルの次元
                 num_key_concepts=16,       # 鍵となる意味概念の数 (アトム数に相当)
                 reduced_dim_per_concept=64, # 各鍵概念の縮減後ベクトル次元
                 model_config=None,
                 module_specific_config=None):
        super().__init__()
        self.input_feature_dim = input_feature_dim
        self.num_key_concepts = module_specific_config.get("num_key_concepts", num_key_concepts)
        self.reduced_dim_per_concept = module_specific_config.get("reduced_dim_per_concept", reduced_dim_per_concept)

        self.key_concept_prototypes = nn.Parameter(
            torch.randn(self.num_key_concepts, self.input_feature_dim)
        )
        nn.init.xavier_uniform_(self.key_concept_prototypes)

        self.reduction_projector = nn.Linear(self.input_feature_dim, self.reduced_dim_per_concept)

        self.final_output_dim = self.num_key_concepts * self.reduced_dim_per_concept

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

    def forward(self, hidden_states_sequence, attention_mask=None, **kwargs):
        # hidden_states_sequence: (batch_size, seq_len, input_feature_dim)

        reduced_concept_vectors_list = []
        # 各鍵概念プロトタイプで入力シーケンス全体から情報を集約
        for i in range(self.num_key_concepts):
            # (batch_size, seq_len, input_feature_dim) と (input_feature_dim)
            concept_specific_attention_scores = torch.matmul(
                hidden_states_sequence, self.key_concept_prototypes[i].unsqueeze(-1)
            ).squeeze(-1) # (batch_size, seq_len)

            if attention_mask is not None: # パディング部分をマスク
                concept_specific_attention_scores = concept_specific_attention_scores.masked_fill(attention_mask == 0, -1e9)

            concept_specific_attention_weights = F.softmax(concept_specific_attention_scores, dim=1) # (batch_size, seq_len)

            # このウェイトで hidden_states を重み付き和 (概念iに関する文脈ベクトル)
            context_vector_for_concept_i = torch.sum(
                hidden_states_sequence * concept_specific_attention_weights.unsqueeze(-1), dim=1
            ) # (batch_size, input_feature_dim)

            reduced_vector_for_concept_i = self.reduction_projector(context_vector_for_concept_i) # (batch_size,
reduced_dim_per_concept)
            reduced_concept_vectors_list.append(reduced_vector_for_concept_i)
```

```
        final_reduced_representation = torch.cat(reduced_concept_vectors_list, dim=1) # (batch_size, num_key_concepts * reduced_dim_per_concept)

        # このモジュールは縮減された固定長ベクトルを返す
        # これが後続の分類器の直接の入力となる
        return {
            "last_hidden_state": final_reduced_representation,
            # "concept_attention_weights": concept_global_attention_weights # (オプション) 全体としてどの概念が活性化したか
        }

    def get_output_dim(self):
        return self.final_output_dim

print("Cell 7.1: InterpretableHolographicReducer class defined.")
```

セル 9

```
# --- セルA: 統合モデルとDebugTrainerの定義 (ホログラフ縮減モジュール対応版) ---

from transformers import PreTrainedModel, AutoModel, AutoConfig, Trainer
from transformers.modeling_outputs import SequenceClassifierOutput
from torch.nn import CrossEntropyLoss
import torch
import torch.nn as nn # nn もここでインポートしておくと確実
import torch.nn.functional as F # F もここでインポートしておくと確実

# 以下のカスタムモジュールクラスが、これより前のセル (セル5, 6, 7, 7.1) で
# 既に定義されていることを前提とします。
# - SemanticConceptModule
# - DirectNonCommutativeModule
# - DiscontinuityEmergenceModule
# - InterpretableHolographicReducer

print("--- Cell A: Defining EnhancedTransformerForSequenceClassification (Holographic Reduction Ready) and DebugTrainer ---")

# --- 統合モデル: EnhancedTransformerForSequenceClassification ---
class EnhancedTransformerForSequenceClassification(PreTrainedModel):
    def __init__(self, hf_config, base_model_name, num_labels, tech_flags=None, module_configs=None):
        super().__init__(hf_config)
        self.num_labels = num_labels
        self.config = hf_config # Hugging Faceのconfigオブジェクトを保存

        # ベースモデルのロードと設定
        self.base_model_prefix = self.config.model_type
        core_model = AutoModel.from_pretrained(base_model_name, config=self.config)
        setattr(self, self.base_model_prefix, core_model)

        current_dim = self.config.hidden_size # ベースモデルの出力次元
        self.tech_modules = nn.ModuleDict()
        self.tech_flags = tech_flags if tech_flags is not None else {}
        self.module_configs = module_configs if module_configs is not None else {}

        # 適用するモジュールの順序を取得 (指定がなければ空リスト)
        self.module_order = self.tech_flags.get("module_order", [])
        self.active_module_names_in_order = [] # 実際に初期化されたモジュールの順序を保持

        # module_order に従ってモジュールを初期化
        for module_key in self.module_order:
            if module_key == "semantic_attention_module" and self.tech_flags.get("use_semantic_attention_module", False):
```

```
            scm_config = self.module_configs.get(module_key, {})
            module_output_dim = scm_config.get("output_dim", current_dim)
            self.tech_modules[module_key] = SemanticConceptModule(
                input_dim=current_dim, output_dim=module_output_dim,
                num_concepts=scm_config.get("num_concepts", 32),
                concept_dim=scm_config.get("concept_dim", 128), model_config=self.config,
                module_specific_config=scm_config # モジュール固有設定を渡す
            )
            current_dim = module_output_dim
            self.active_module_names_in_order.append(module_key)
            print(f"{module_key} enabled and initialized. Output dim: {current_dim}")

        elif module_key == "direct_non_commutative_module" and self.tech_flags.get("use_direct_non_commutative_module",
False):
            dnc_config = self.module_configs.get(module_key, {})
            module_output_dim = dnc_config.get("output_dim", current_dim)
            self.tech_modules[module_key] = DirectNonCommutativeModule(
                input_dim=current_dim, output_dim=module_output_dim,
                dropout=dnc_config.get("dropout", 0.2), model_config=self.config,
                module_specific_config=dnc_config
            )
            current_dim = module_output_dim
            self.active_module_names_in_order.append(module_key)
            print(f"{module_key} enabled and initialized. Output dim: {current_dim}")

        elif module_key == "discontinuity_emergence_module" and self.tech_flags.get("use_discontinuity_emergence_module",
False):
            dem_config = self.module_configs.get(module_key, {})
            module_output_dim = dem_config.get("output_dim", current_dim)
            self.tech_modules[module_key] = DiscontinuityEmergenceModule(
                input_dim=current_dim, output_dim=module_output_dim,
                semantic_dim_for_discontinuity=dem_config.get("semantic_dim_for_discontinuity", 12),
                emergent_label_dim=dem_config.get("emergent_label_dim", 24),
                dropout=dem_config.get("dropout", 0.2), model_config=self.config,
                module_specific_config=dem_config
            )
            current_dim = module_output_dim
            self.active_module_names_in_order.append(module_key)
            print(f"{module_key} enabled and initialized. Output dim: {current_dim}")

        elif module_key == "holographic_reduction_module" and self.tech_flags.get("use_holographic_reduction_module",
False):
            hr_config = self.module_configs.get(module_key, {})
            # InterpretableHolographicReducer は自身のget_output_dim()で出力次元を返す
            # input_feature_dim は current_dim (前の層の出力次元)
            self.tech_modules[module_key] = InterpretableHolographicReducer(
                input_feature_dim=current_dim,
                # num_key_concepts と reduced_dim_per_concept は module_specific_config から渡す
                module_specific_config=hr_config,
                model_config=self.config
            )
            current_dim = self.tech_modules[module_key].get_output_dim() # 縮減後の次元に更新
            self.active_module_names_in_order.append(module_key)
            print(f"{module_key} enabled and initialized. Output dim (classifier input): {current_dim}")

        # 他のモジュールも同様に追加可能

    self.dropout = nn.Dropout(self.config.hidden_dropout_prob if hasattr(self.config, 'hidden_dropout_prob') else 0.1)
    self.classifier = nn.Linear(current_dim, self.num_labels) # 最終的なcurrent_dimで分類器を初期化
    print(f"EnhancedTransformerForSequenceClassification initialized. Final classifier input dim: {current_dim}.")
    if self.active_module_names_in_order:
```

```
            print(f"  Applied modules (in order): {self.active_module_names_in_order}")
        else:
            print("  No additional tech modules applied (Baseline configuration).")


    def forward(
        self, input_ids=None, attention_mask=None, token_type_ids=None, labels=None,
        return_dict=None, **kwargs ): # target_semantic_labels は kwargs に含まれる想定もし必要なら

        # print(f"--- EnhancedTransformer.forward CALLED (is_training: {self.training}) ---")
        # if labels is not None: print(f"  Forward RECEIVED labels, shape: {labels.shape}")
        # else: print(f"  Forward did NOT receive labels this call.")

        base_model_kwargs = { k: v for k, v in kwargs.items() if k in ["position_ids", "head_mask", "inputs_embeds",
"output_attentions", "output_hidden_states"]}
        transformer_outputs = self.base_model(input_ids=input_ids, attention_mask=attention_mask,
token_type_ids=token_type_ids, return_dict=True, **base_model_kwargs)
        current_features = transformer_outputs.last_hidden_state # (batch, seq_len, hidden_dim)

        all_module_specific_outputs = {}

        # module_order に従ってモジュールを順次適用
        for module_name_key in self.active_module_names_in_order: # __init__で実際に初期化された順序リストを使用
            if module_name_key in self.tech_modules:
                # print(f"  Applying module: {module_name_key}")
                # HolographicReductionModule はシーケンスを受け取り、固定長ベクトルを返す想定
                # それ以外のモジュールはシーケンスを受け取り、シーケンスを返す想定
                # この分岐は HolographicReductionModule が常に module_order の最後に来るという前提に基づく
                # もし途中に来る場合は、current_features の形状が変わるため、より複雑な制御が必要

                module_output_dict = self.tech_modules[module_name_key](current_features, attention_mask=attention_mask)
                current_features = module_output_dict["last_hidden_state"]

                for output_key, output_value in module_output_dict.items():
                    if output_key != "last_hidden_state":
                        all_module_specific_outputs[f"{module_name_key}_{output_key}"] = output_value

        # プーリング処理
        # HolographicReductionModule が適用された場合、current_features は既に (batch, reduced_dim) のはず
        # そうでない場合は、通常のCLSトークンなどによるプーリングを行う
        is_holographic_active_and_last = self.active_module_names_in_order and \
                                         self.active_module_names_in_order[-1] == "holographic_reduction_module" and \
                                         "holographic_reduction_module" in self.tech_modules

        if is_holographic_active_and_last:
            pooled_output = current_features # ホログラフ縮減モジュールの出力 (既にプーリング済み)
            # print(f"  Using Holographic Reduction output as pooled_output. Shape: {pooled_output.shape}")
        else:
            if current_features.ndim == 3: # (batch, seq, dim)
                pooled_output = current_features[:, 0] # [CLS] トークンを使用 (または他のプーリング)
                # print(f"  Using CLS token pooling. Shape: {pooled_output.shape}")
            elif current_features.ndim == 2: # 既にプーリング済みの場合 (例: 前のモジュールがプーリングした場合)
                pooled_output = current_features
                # print(f"  Input 'current_features' is already pooled. Shape: {pooled_output.shape}")
            else:
                raise ValueError(f"Unsupported shape for current_features before classifier: {current_features.shape}")

        pooled_output = self.dropout(pooled_output)
        logits = self.classifier(pooled_output)

        loss = None
```

```
        if labels is not None:
            loss_fct = CrossEntropyLoss()
            loss = loss_fct(logits.view(-1, self.num_labels), labels.view(-1))
            # if loss is not None: print(f"  Forward CALCULATED loss: {loss.item()}")

        # print(f"--- EnhancedTransformer.forward RETURNING ... ---")

        final_outputs_tuple = (logits,)
        # ベースモデルの主要な出力と、モジュールからの補助的な出力を選択的に含める
        # (今回はシンプルにベースモデルの主要なものと、モジュールからの全追加出力を返す)
        if hasattr(transformer_outputs, 'hidden_states') and transformer_outputs.hidden_states is not None:
            final_outputs_tuple += (transformer_outputs.hidden_states,)
        if hasattr(transformer_outputs, 'attentions') and transformer_outputs.attentions is not None:
            final_outputs_tuple += (transformer_outputs.attentions,)
        if all_module_specific_outputs:
            final_outputs_tuple += (all_module_specific_outputs,)

        if loss is not None:
            return (loss,) + final_outputs_tuple
        else:
            return (None,) + final_outputs_tuple

# --- DebugTrainer クラスの定義 (変更なし) ---
class DebugTrainer(Trainer):
    def prediction_step(
        self, model: nn.Module, inputs: dict[str, torch.Tensor | nn.utils.rnn.PackedSequence],
        prediction_loss_only: bool, ignore_keys: list[str] | None = None,
    ) -> tuple[torch.Tensor | None, torch.Tensor | None, torch.Tensor | None]:
        # print(f"--- DebugTrainer.prediction_step CALLED ---")
        model.eval()
        with torch.no_grad(): model_outputs = model(**inputs)
        loss = model_outputs[0] if model_outputs[0] is not None else None
        logits = model_outputs[1] if len(model_outputs) > 1 and model_outputs[1] is not None else None
        labels_out = inputs.get("labels")
        if prediction_loss_only: return (loss, None, None)
        return loss, logits, labels_out

print("Cell A: EnhancedTransformerForSequenceClassification (Holographic Reduction capable) and DebugTrainer defined.")
```

セル１０

```
# --- セルB: 実験構成、グローバル変数パック、共通実行関数 (完全統合版) ---

print("--- Cell B (Fully Integrated): Defining All Experiment Setups and Utilities ---")

# 1. 前提となるグローバル変数の確認
required_globals_cell_B_full = [
    'hf_model_config', 'BASE_MODEL_NAME', 'NUM_LABELS', 'OUTPUT_DIR_BASE',
    'LEARNING_RATE', 'BATCH_SIZE', 'NUM_EPOCHS',
    'EnhancedTransformerForSequenceClassification',
    'train_dataset', 'eval_dataset', 'tokenizer', 'compute_metrics',
    'DebugTrainer', 'Dataset',
    'SemanticConceptModule', 'DirectNonCommutativeModule',
    'DiscontinuityEmergenceModule', 'InterpretableHolographicReducer'
]
missing_globals_check_B_full = False
for var_name in required_globals_cell_B_full:
    if var_name not in globals():
        print(f"ERROR in Cell B (Fully Integrated): Global variable or class '{var_name}' is not defined!")
        missing_globals_check_B_full = True
```

```
if missing_globals_check_B_full:
    raise NameError("One or more required global variables/classes for Cell B (Fully Integrated) are not defined. Please check prerequisite cells (1 through A).")
else:
    print("All prerequisite global variables and classes for Cell B (Fully Integrated) seem to be defined.")

# 2. 実験構成の定義 (お客様の全パターン + ホログラフ縮減テスト)
experiment_configurations = [
    # --- ベースライン ---
    {"name": "E0_Baseline", "tech_flags": {"module_order": []}, "module_configs": {}, "#日本語": "ベースライン"},
    # --- 単一モジュール ---
    {"name": "E1_SemanticAttention",
     "tech_flags": {"module_order": ["semantic_attention_module"], "use_semantic_attention_module": True},
     "module_configs": {"semantic_attention_module": {"num_concepts": 32, "concept_dim": 128, "output_dim": hf_model_config.hidden_size}}, "#日本語": "意味アテンション"},
    {"name": "E2_DirectNonCommutative",
     "tech_flags": {"module_order": ["direct_non_commutative_module"], "use_direct_non_commutative_module": True},
     "module_configs": {"direct_non_commutative_module": {"output_dim": hf_model_config.hidden_size, "dropout": 0.2, "internal_processing_dim": hf_model_config.hidden_size}}, "#日本語": "非可換"},
    {"name": "E3_DiscontinuityEmergence",
     "tech_flags": {"module_order": ["discontinuity_emergence_module"], "use_discontinuity_emergence_module": True},
     "module_configs": {"discontinuity_emergence_module": {"output_dim": hf_model_config.hidden_size, "semantic_dim_for_discontinuity": 12, "emergent_label_dim": 24, "dropout": 0.2}}, "#日本語": "断絶創発"},

    # --- 2モジュールの組み合わせ (順序考慮) ---
    {"name": "E4_SemAtt_then_NonComm",
     "tech_flags": {"module_order": ["semantic_attention_module", "direct_non_commutative_module"], "use_semantic_attention_module": True, "use_direct_non_commutative_module": True},
     "module_configs": {"semantic_attention_module": {"num_concepts": 32, "concept_dim": 128, "output_dim": hf_model_config.hidden_size},
                        "direct_non_commutative_module": {"output_dim": hf_model_config.hidden_size, "dropout": 0.2, "internal_processing_dim": hf_model_config.hidden_size}}, "#日本語": "意味アテンション → 非可換"},
    {"name": "E5_NonComm_then_SemAtt",
     "tech_flags": {"module_order": ["direct_non_commutative_module", "semantic_attention_module"], "use_direct_non_commutative_module": True, "use_semantic_attention_module": True},
     "module_configs": {"direct_non_commutative_module": {"output_dim": hf_model_config.hidden_size, "dropout": 0.2, "internal_processing_dim": hf_model_config.hidden_size},
                        "semantic_attention_module": {"num_concepts": 32, "concept_dim": 128, "output_dim": hf_model_config.hidden_size}}, "#日本語": "非可換 → 意味アテンション"},
    {"name": "E6_SemAtt_then_DiscEm",
     "tech_flags": {"module_order": ["semantic_attention_module", "discontinuity_emergence_module"], "use_semantic_attention_module": True, "use_discontinuity_emergence_module": True},
     "module_configs": {"semantic_attention_module": {"num_concepts": 32, "concept_dim": 128, "output_dim": hf_model_config.hidden_size},
                        "discontinuity_emergence_module": {"output_dim": hf_model_config.hidden_size, "semantic_dim_for_discontinuity": 12, "emergent_label_dim": 24, "dropout": 0.2}}, "#日本語": "意味アテンション → 断絶創発"},
    {"name": "E7_DiscEm_then_SemAtt",
     "tech_flags": {"module_order": ["discontinuity_emergence_module", "semantic_attention_module"], "use_discontinuity_emergence_module": True, "use_semantic_attention_module": True},
     "module_configs": {"discontinuity_emergence_module": {"output_dim": hf_model_config.hidden_size, "semantic_dim_for_discontinuity": 12, "emergent_label_dim": 24, "dropout": 0.2},
                        "semantic_attention_module": {"num_concepts": 32, "concept_dim": 128, "output_dim": hf_model_config.hidden_size}}, "#日本語": "断絶創発 → 意味アテンション"},
    {"name": "E8_NonComm_then_DiscEm",
     "tech_flags": {"module_order": ["direct_non_commutative_module", "discontinuity_emergence_module"], "use_direct_non_commutative_module": True, "use_discontinuity_emergence_module": True},
     "module_configs": {"direct_non_commutative_module": {"output_dim": hf_model_config.hidden_size, "dropout": 0.2, "internal_processing_dim": hf_model_config.hidden_size},
                        "discontinuity_emergence_module": {"output_dim": hf_model_config.hidden_size, "semantic_dim_for_discontinuity": 12, "emergent_label_dim": 24, "dropout": 0.2}}, "#日本語": "非可換 → 断絶創発"},
    {"name": "E9_DiscEm_then_NonComm",
```

```
    "tech_flags": {"module_order": ["discontinuity_emergence_module", "direct_non_commutative_module"],
"use_discontinuity_emergence_module": True, "use_direct_non_commutative_module": True},
    "module_configs": {"discontinuity_emergence_module": {"output_dim": hf_model_config.hidden_size,
"semantic_dim_for_discontinuity": 12, "emergent_label_dim": 24, "dropout": 0.2},
                       "direct_non_commutative_module": {"output_dim": hf_model_config.hidden_size, "dropout": 0.2,
"internal_processing_dim": hf_model_config.hidden_size}}, "#日本語": "断絶創発 → 非可換"},

   # --- 3モジュールの組み合わせ ---
   {"name": "E10_SemAtt_NonComm_DiscEm",
    "tech_flags": {"module_order": ["semantic_attention_module", "direct_non_commutative_module",
"discontinuity_emergence_module"], "use_semantic_attention_module": True, "use_direct_non_commutative_module": True,
"use_discontinuity_emergence_module": True},
    "module_configs": {
        "semantic_attention_module": {"num_concepts": 32, "concept_dim": 128, "output_dim": hf_model_config.hidden_size},
        "direct_non_commutative_module": {"output_dim": hf_model_config.hidden_size, "dropout": 0.2,
"internal_processing_dim": hf_model_config.hidden_size},
        "discontinuity_emergence_module": {"output_dim": hf_model_config.hidden_size, "semantic_dim_for_discontinuity": 12,
"emergent_label_dim": 24, "dropout": 0.2}}, "#日本語": "意味アテンション → 非可換 → 断絶創発"},
   {"name": "E11_NonComm_SemAtt_DiscEm",
    "tech_flags": {"module_order": ["direct_non_commutative_module", "semantic_attention_module",
"discontinuity_emergence_module"], "use_direct_non_commutative_module": True, "use_semantic_attention_module": True,
"use_discontinuity_emergence_module": True},
    "module_configs": {"direct_non_commutative_module": {"output_dim": hf_model_config.hidden_size, "dropout": 0.2,
"internal_processing_dim": hf_model_config.hidden_size},
                       "semantic_attention_module": {"num_concepts": 32, "concept_dim": 128, "output_dim":
hf_model_config.hidden_size},
                       "discontinuity_emergence_module": {"output_dim": hf_model_config.hidden_size,
"semantic_dim_for_discontinuity": 12, "emergent_label_dim": 24, "dropout": 0.2}}, "#日本語": "非可換 → 意味アテンション → 断絶
創発"},
   {"name": "E12_DiscEm_SemAtt_NonComm",
    "tech_flags": {"module_order": ["discontinuity_emergence_module", "semantic_attention_module",
"direct_non_commutative_module"], "use_discontinuity_emergence_module": True, "use_semantic_attention_module": True,
"use_direct_non_commutative_module": True},
    "module_configs": {"discontinuity_emergence_module": {"output_dim": hf_model_config.hidden_size,
"semantic_dim_for_discontinuity": 12, "emergent_label_dim": 24, "dropout": 0.2},
                       "semantic_attention_module": {"num_concepts": 32, "concept_dim": 128, "output_dim":
hf_model_config.hidden_size},
                       "direct_non_commutative_module": {"output_dim": hf_model_config.hidden_size, "dropout": 0.2,
"internal_processing_dim": hf_model_config.hidden_size}}, "#日本語": "断絶創発 → 意味アテンション → 非可換"},
   {"name": "E13_SemAtt_DiscEm_NonComm",
    "tech_flags": {"module_order": ["semantic_attention_module", "discontinuity_emergence_module",
"direct_non_commutative_module"], "use_semantic_attention_module": True, "use_discontinuity_emergence_module": True,
"use_direct_non_commutative_module": True},
    "module_configs": {"semantic_attention_module": {"num_concepts": 32, "concept_dim": 128, "output_dim":
hf_model_config.hidden_size},
                       "discontinuity_emergence_module": {"output_dim": hf_model_config.hidden_size,
"semantic_dim_for_discontinuity": 12, "emergent_label_dim": 24, "dropout": 0.2},
                       "direct_non_commutative_module": {"output_dim": hf_model_config.hidden_size, "dropout": 0.2,
"internal_processing_dim": hf_model_config.hidden_size}}, "#日本語": "意味アテンション → 断絶創発 → 非可換"},
   {"name": "E14_NonComm_DiscEm_SemAtt",
    "tech_flags": {"module_order": ["direct_non_commutative_module", "discontinuity_emergence_module",
"semantic_attention_module"], "use_direct_non_commutative_module": True, "use_discontinuity_emergence_module": True,
"use_semantic_attention_module": True},
    "module_configs": {"direct_non_commutative_module": {"output_dim": hf_model_config.hidden_size, "dropout": 0.2,
"internal_processing_dim": hf_model_config.hidden_size},
                       "discontinuity_emergence_module": {"output_dim": hf_model_config.hidden_size,
"semantic_dim_for_discontinuity": 12, "emergent_label_dim": 24, "dropout": 0.2},
                       "semantic_attention_module": {"num_concepts": 32, "concept_dim": 128, "output_dim":
hf_model_config.hidden_size}}, "#日本語": "非可換 → 断絶創発 → 意味アテンション"},
   {"name": "E15_DiscEm_NonComm_SemAtt",
```

```
     "tech_flags": {"module_order": ["discontinuity_emergence_module", "direct_non_commutative_module",
"semantic_attention_module"], "use_discontinuity_emergence_module": True, "use_direct_non_commutative_module": True,
"use_semantic_attention_module": True},
     "module_configs": {"discontinuity_emergence_module": {"output_dim": hf_model_config.hidden_size,
"semantic_dim_for_discontinuity": 12, "emergent_label_dim": 24, "dropout": 0.2},
                        "direct_non_commutative_module": {"output_dim": hf_model_config.hidden_size, "dropout": 0.2,
"internal_processing_dim": hf_model_config.hidden_size},
                        "semantic_attention_module": {"num_concepts": 32, "concept_dim": 128, "output_dim":
hf_model_config.hidden_size}}, "#日本語": "断絶創発 → 非可換 → 意味アテンション"},

    # --- ホログラフ縮減モジュールを含むテスト構成 ---
    {"name": "E16_Baseline_then_HolographicReduction",
     "tech_flags": {"module_order": ["holographic_reduction_module"], "use_holographic_reduction_module": True},
     "module_configs": {"holographic_reduction_module": {"num_key_concepts": 16, "reduced_dim_per_concept": 32,
"reduction_type": "attention_pooling"}}, "#日本語": "ベースライン → ホログラフ縮減(AttPool)"},
    {"name": "E17_E6_then_HolographicReduction",
     "tech_flags": {"module_order": ["semantic_attention_module", "direct_non_commutative_module",
"discontinuity_emergence_module", "holographic_reduction_module"],
                    "use_semantic_attention_module": True, "use_direct_non_commutative_module": True,
"use_discontinuity_emergence_module": True, "use_holographic_reduction_module": True},
     "module_configs": {
         "semantic_attention_module": {"num_concepts": 32, "concept_dim": 128, "output_dim": hf_model_config.hidden_size},
         "direct_non_commutative_module": {"output_dim": hf_model_config.hidden_size, "dropout": 0.2,
"internal_processing_dim": hf_model_config.hidden_size},
         "discontinuity_emergence_module": {"output_dim": hf_model_config.hidden_size, "semantic_dim_for_discontinuity": 12,
"emergent_label_dim": 24, "dropout": 0.2},
         "holographic_reduction_module": {"num_key_concepts": 16, "reduced_dim_per_concept": 32, "reduction_type":
"attention_pooling"}}, "#日本語": "意味Att→非可換→断絶創発→ホログラフ(AttPool)"},
]
print(f"Total experiment configurations defined in Cell B (Fully Integrated): {len(experiment_configurations)}")

# 3. グローバル変数のパッケージ化
global_variables_for_experiments = {
    'hf_model_config': hf_model_config, 'BASE_MODEL_NAME': BASE_MODEL_NAME, 'NUM_LABELS': NUM_LABELS,
    'OUTPUT_DIR_BASE': OUTPUT_DIR_BASE, 'LEARNING_RATE': LEARNING_RATE, 'BATCH_SIZE': BATCH_SIZE,
    'NUM_EPOCHS': NUM_EPOCHS,
    'EnhancedTransformerForSequenceClassification': EnhancedTransformerForSequenceClassification,
    'train_dataset': train_dataset, 'eval_dataset': eval_dataset, 'tokenizer': tokenizer,
    'compute_metrics': compute_metrics, 'DebugTrainer': DebugTrainer, 'Dataset': Dataset,
    'SemanticConceptModule': SemanticConceptModule,
    'DirectNonCommutativeModule': DirectNonCommutativeModule,
    'DiscontinuityEmergenceModule': DiscontinuityEmergenceModule,
    'InterpretableHolographicReducer': InterpretableHolographicReducer
}
print("Global variables for experiments packaged.")

# 4. 共通実験実行関数 (run_ablation_experiment)
def run_ablation_experiment(exp_config, global_vars_dict):
    config_name = exp_config["name"]
    tech_flags = exp_config["tech_flags"]
    module_cfgs = exp_config["module_configs"]

    print(f"\n--- Running Experiment: {config_name} ---")
    print(f"  Technology Flags: {tech_flags}")
    print(f"  Module Configs: {module_cfgs}")

    hf_cfg = global_vars_dict['hf_model_config']
    base_model_name_local = global_vars_dict['BASE_MODEL_NAME']
    n_labels = global_vars_dict['NUM_LABELS']
    output_dir_base_local = global_vars_dict['OUTPUT_DIR_BASE']
    lr = global_vars_dict['LEARNING_RATE']
```

```
    bs = global_vars_dict['BATCH_SIZE']
    n_epochs = global_vars_dict['NUM_EPOCHS']

    enh_transformer_class_local = global_vars_dict['EnhancedTransformerForSequenceClassification']
    tr_dataset = global_vars_dict['train_dataset']
    ev_dataset = global_vars_dict['eval_dataset']
    tok = global_vars_dict['tokenizer']
    comp_metrics = global_vars_dict['compute_metrics']
    dbg_trainer_class_local = global_vars_dict['DebugTrainer']
    dataset_class_local = global_vars_dict['Dataset']

    if ev_dataset is not None and isinstance(ev_dataset, dataset_class_local):
        if 'labels' not in ev_dataset.column_names:
            print(f"    WARNING (inside run_ablation_experiment for {config_name}): 'labels' column not found in eval_dataset!")
    else:
        print(f"  Warning (inside run_ablation_experiment for {config_name}): eval_dataset not found or not a Dataset object.")

    model = enh_transformer_class_local(
        hf_config=hf_cfg, base_model_name=base_model_name_local, num_labels=n_labels,
        tech_flags=tech_flags, module_configs=module_cfgs
    )
    current_output_dir = f"{output_dir_base_local}{config_name}"

    training_args = TrainingArguments(
        output_dir=current_output_dir, learning_rate=lr, per_device_train_batch_size=bs,
        per_device_eval_batch_size=bs, num_train_epochs=n_epochs, weight_decay=0.01,
        evaluation_strategy="epoch", logging_strategy="epoch", save_strategy="epoch",
        load_best_model_at_end=True, metric_for_best_model="accuracy", report_to="none",
        do_eval=True, remove_unused_columns=False,
        # save_total_limit=1, # ディスク容量を節約する場合
    )
    trainer = dbg_trainer_class_local(
        model=model, args=training_args, train_dataset=tr_dataset, eval_dataset=ev_dataset,
        tokenizer=tok, compute_metrics=comp_metrics,
    )
    print(f"Starting training for {config_name}...")
    eval_results_dict = {}
    try:
        trainer.train()
        print(f"Training finished for {config_name}.")
        print(f"Starting final evaluation for {config_name} (with best model)...")
        eval_results = trainer.evaluate()
        print(f"Evaluation results for {config_name}: {eval_results}")
        eval_results_dict = eval_results
    except Exception as e:
        print(f"!!! An error occurred during training or evaluation for {config_name}: {e} !!!")
        import traceback
        traceback.print_exc()
        eval_results_dict = {"error": str(e), "eval_accuracy": float('nan'), "eval_loss": float('nan')}
    return config_name, eval_results_dict

# 5. 全ての実験結果を格納する辞書の初期化
all_experiment_results_summary = {}

print("\nCell B (Fully Integrated): All setups for running experiments are complete.")
print(f"To run experiments, execute the individual experiment cells (e.g., Cell_E0, Cell_E1, etc.) one by one.")
```

実験コード

E0

```
# E0--- ベースライン ---
print("--- Executing Experiment E2_DirectNonCommutative (Baseline + Direct Non-Commutative) ---")

exp_to_run_name_e2 = "E2_DirectNonCommutative"
config_to_run_e2 = next((config for config in experiment_configurations if config["name"] == exp_to_run_name_e2), None)

if config_to_run_e2:
    if 'all_experiment_results_summary' not in globals(): all_experiment_results_summary = {}
    if 'global_variables_for_experiments' not in globals(): raise NameError("global_variables_for_experiments is not defined.")

    name, result = run_ablation_experiment(config_to_run_e2, global_variables_for_experiments)
    all_experiment_results_summary[name] = result
    print(f"\n--- Results for {name} ---")
    print(result)
else:
    print(f"Configuration for '{exp_to_run_name_e2}' not found.")

print(f"--- Finished Experiment E2_DirectNonCommutative ---")
```

E1

```
# E1---意味ラベル
print("--- Executing Experiment E1_SemanticAttention (Baseline + Semantic Attention) ---")

exp_to_run_name_e1 = "E1_SemanticAttention"
config_to_run_e1 = next((config for config in experiment_configurations if config["name"] == exp_to_run_name_e1), None)

if config_to_run_e1:
    if 'all_experiment_results_summary' not in globals(): all_experiment_results_summary = {}
    if 'global_variables_for_experiments' not in globals(): raise NameError("global_variables_for_experiments is not defined.")

    name, result = run_ablation_experiment(config_to_run_e1, global_variables_for_experiments)
    all_experiment_results_summary[name] = result
    print(f"\n--- Results for {name} ---")
    print(result)
else:
    print(f"Configuration for '{exp_to_run_name_e1}' not found.")

print(f"--- Finished Experiment E1_SemanticAttention ---")
```

E2

```
# E2 --- 非可換
print("--- Executing Experiment E5_NonComm_then_SemAtt ---")

exp_to_run_name_e5 = "E5_NonComm_then_SemAtt"
config_to_run_e5 = next((config for config in experiment_configurations if config["name"] == exp_to_run_name_e5), None)

if config_to_run_e5:
    # ... (run_ablation_experiment の呼び出しと結果格納は同様) ...
    name, result = run_ablation_experiment(config_to_run_e5, global_variables_for_experiments)
    all_experiment_results_summary[name] = result
    print(f"\n--- Results for {name} ---")
    print(result)
else:
    print(f"Configuration for '{exp_to_run_name_e5}' not found.")
```

```
print(f"--- Finished Experiment E5_NonComm_then_SemAtt ---")
```

E3

```
# E3 ---断絶創発
print("--- Executing Experiment E3_DiscontinuityEmergence (Baseline + Discontinuity Emergence) ---")

exp_to_run_name_e3 = "E3_DiscontinuityEmergence"
config_to_run_e3 = next((config for config in experiment_configurations if config["name"] == exp_to_run_name_e3), None)

if config_to_run_e3:
    if 'all_experiment_results_summary' not in globals(): all_experiment_results_summary = {}
    if 'global_variables_for_experiments' not in globals(): raise NameError("global_variables_for_experiments is not defined.")

    name, result = run_ablation_experiment(config_to_run_e3, global_variables_for_experiments)
    all_experiment_results_summary[name] = result
    print(f"\n--- Results for {name} ---")
    print(result)
else:
    print(f"Configuration for '{exp_to_run_name_e3}' not found.")

print(f"--- Finished Experiment E3_DiscontinuityEmergence ---")
```

E4

```
# E4--- 意味ラベル＋非可換　--
print("--- Executing Experiment E13_SemAtt_DiscEm_NonComm ---")
exp_to_run_name_e13 = "E13_SemAtt_DiscEm_NonComm"
config_to_run_e13 = next((config for config in experiment_configurations if config["name"] == exp_to_run_name_e13), None)

if config_to_run_e13:
    name, result = run_ablation_experiment(config_to_run_e13, global_variables_for_experiments)
    all_experiment_results_summary[name] = result
    print(f"\n--- Results for {name} ---")
    print(result)
else:
    print(f"Configuration for '{exp_to_run_name_e13}' not found.")
print(f"--- Finished Experiment E13_SemAtt_DiscEm_NonComm ---")
```

E 5

```
# E5 --- 非可換＋意味ラベル ---
print("--- Executing Experiment E5_NonComm_then_SemAtt ---")

exp_to_run_name_e5 = "E5_NonComm_then_SemAtt"
config_to_run_e5 = next((config for config in experiment_configurations if config["name"] == exp_to_run_name_e5), None)

if config_to_run_e5:
    if 'all_experiment_results_summary' not in globals(): all_experiment_results_summary = {}
    if 'global_variables_for_experiments' not in globals(): raise NameError("global_variables_for_experiments is not defined.")

    name, result = run_ablation_experiment(config_to_run_e5, global_variables_for_experiments)
    all_experiment_results_summary[name] = result # 結果をグローバル辞書に格納
    print(f"\n--- Results for {name} ---")
    print(result)
else:
    print(f"Configuration for '{exp_to_run_name_e5}' not found in experiment_configurations.")
```

```
print(f"--- Finished Experiment E5_NonComm_then_SemAtt ---")
```

E 6

```
# E6--- 意味ラベル＋断絶創発---
print("--- Executing Experiment E6_SemAtt_then_DiscEm ---")
exp_to_run_name_e6 = "E6_SemAtt_then_DiscEm"
config_to_run_e6 = next((config for config in experiment_configurations if config["name"] == exp_to_run_name_e6), None)

if config_to_run_e6:
    name, result = run_ablation_experiment(config_to_run_e6, global_variables_for_experiments)
    all_experiment_results_summary[name] = result
    print(f"\n--- Results for {name} ---")
    print(result)
else:
    print(f"Configuration for '{exp_to_run_name_e6}' not found.")
print(f"--- Finished Experiment E9_SemAtt_then_DiscEm ---")
```

E 7

```
#E7 --- 断絶創発＋意味ラベル ---
print("--- Executing Experiment E7_DiscEm_then_SemAtt ---")

exp_to_run_name_e7 = "E7_DiscEm_then_SemAtt"
config_to_run_e7 = next((config for config in experiment_configurations if config["name"] == exp_to_run_name_e7), None)

if config_to_run_e7:
    if 'all_experiment_results_summary' not in globals(): all_experiment_results_summary = {}
    if 'global_variables_for_experiments' not in globals(): raise NameError("global_variables_for_experiments is not defined.")

    name, result = run_ablation_experiment(config_to_run_e7, global_variables_for_experiments)
    all_experiment_results_summary[name] = result # 結果をグローバル辞書に格納
    print(f"\n--- Results for {name} ---")
    print(result)
else:
    print(f"Configuration for '{exp_to_run_name_e7}' not found in experiment_configurations.")

print(f"--- Finished Experiment E7_DiscEm_then_SemAtt ---")
```

E 8

```
# E8 ---非可換＋断絶創発  ---
print("--- Executing Experiment E8_NonComm_then_DiscEm ---")

exp_to_run_name_e8 = "E8_NonComm_then_DiscEm"
config_to_run_e8 = next((config for config in experiment_configurations if config["name"] == exp_to_run_name_e8), None)

if config_to_run_e8:
    if 'all_experiment_results_summary' not in globals(): all_experiment_results_summary = {}
    if 'global_variables_for_experiments' not in globals(): raise NameError("global_variables_for_experiments is not defined.")

    name, result = run_ablation_experiment(config_to_run_e8, global_variables_for_experiments)
    all_experiment_results_summary[name] = result # 結果をグローバル辞書に格納
    print(f"\n--- Results for {name} ---")
    print(result)
else:
```

```
    print(f"Configuration for '{exp_to_run_name_e8}' not found in experiment_configurations.")

print(f"--- Finished Experiment E11_NonComm_then_DiscEm ---")
```

E９

```
# E9 --- 断絶創発＋非可換 ---
print("--- Executing Experiment E9_DiscEm_then_NonComm ---")

exp_to_run_name_e9 = "E9_DiscEm_then_NonComm"
config_to_run_e9 = next((config for config in experiment_configurations if config["name"] == exp_to_run_name_e9), None)

if config_to_run_e9:
    if 'all_experiment_results_summary' not in globals(): all_experiment_results_summary = {}
    if 'global_variables_for_experiments' not in globals(): raise NameError("global_variables_for_experiments is not defined.")

    name, result = run_ablation_experiment(config_to_run_e9, global_variables_for_experiments)
    all_experiment_results_summary[name] = result # 結果をグローバル辞書に格納
    print(f"\n--- Results for {name} ---")
    print(result)
else:
    print(f"Configuration for '{exp_to_run_name_e9}' not found in experiment_configurations.")

print(f"--- Finished Experiment E9_DiscEm_then_NonComm ---")
```

E１０

```
# E10--- 意味アテンション＋非可換＋断絶創発
print("--- Executing Experiment E10_SemAtt_NonComm_DiscEm ---")
exp_to_run_name_e10 = "E10_SemAtt_NonComm_DiscEm"
config_to_run_e10 = next((config for config in experiment_configurations if config["name"] == exp_to_run_name_e10), None)

if config_to_run_e10:
    # ... (run_ablation_experiment の呼び出しと結果格納は同様) ...
    name, result = run_ablation_experiment(config_to_run_e10, global_variables_for_experiments)
    all_experiment_results_summary[name] = result
    print(f"\n--- Results for {name} ---")
    print(result)
else:
    print(f"Configuration for '{exp_to_run_name_e10}' not found.")
print(f"--- Finished Experiment E6_SemAtt_NonComm_DiscEm ---")
```

E１１

```
# E11---非可換＋意味アテンション＋断絶創発
print("--- Executing Experiment E11_NonComm_SemAtt_DiscEm ---")

exp_to_run_name_e11 = "E11_NonComm_SemAtt_DiscEm"
config_to_run_e11 = next((config for config in experiment_configurations if config["name"] == exp_to_run_name_e7), None)

if config_to_run_e11:
    if 'all_experiment_results_summary' not in globals(): all_experiment_results_summary = {}
    if 'global_variables_for_experiments' not in globals(): raise NameError("global_variables_for_experiments is not defined.")

    name, result = run_ablation_experiment(config_to_run_e11, global_variables_for_experiments)
    all_experiment_results_summary[name] = result # 結果をグローバル辞書に格納
```

```
    print(f"\n--- Results for {name} ---")
    print(result)
else:
    print(f"Configuration for '{exp_to_run_name_e11}' not found in experiment_configurations.")

print(f"--- Finished Experiment E11_NonComm_SemAtt_DiscEm ---")
```

E１２

```
# E12 --- 断絶創発＋意味ラベル＋非可換
print("--- Executing Experiment E12_DiscEm_SemAtt_NonComm ---")

exp_to_run_name_e12 = "E12_DiscEm_SemAtt_NonComm"
config_to_run_e12 = next((config for config in experiment_configurations if config["name"] == exp_to_run_name_e12), None)

if config_to_run_e12:
    if 'all_experiment_results_summary' not in globals(): all_experiment_results_summary = {}
    if 'global_variables_for_experiments' not in globals(): raise NameError("global_variables_for_experiments is not defined.")

    name, result = run_ablation_experiment(config_to_run_e12, global_variables_for_experiments)
    all_experiment_results_summary[name] = result # 結果をグローバル辞書に格納
    print(f"\n--- Results for {name} ---")
    print(result)
else:
    print(f"Configuration for '{exp_to_run_name_e12}' not found in experiment_configurations.")

print(f"--- Finished Experiment E12_DiscEm_SemAtt_NonComm ---")
```

E１３

```
# E13---意味ラベル＋断絶創発＋非可換
print("--- Executing Experiment E13_SemAtt_DiscEm_NonComm ---")

exp_to_run_name_e13 = "E13_SemAtt_DiscEm_NonComm"
config_to_run_e13 = next((config for config in experiment_configurations if config["name"] == exp_to_run_name_e13), None)

if config_to_run_e13:
    if 'all_experiment_results_summary' not in globals(): all_experiment_results_summary = {}
    if 'global_variables_for_experiments' not in globals(): raise NameError("global_variables_for_experiments is not defined.")

    name, result = run_ablation_experiment(config_to_run_e13, global_variables_for_experiments)
    all_experiment_results_summary[name] = result # 結果をグローバル辞書に格納
    print(f"\n--- Results for {name} ---")
    print(result)
else:
    print(f"Configuration for '{exp_to_run_name_e13}' not found in experiment_configurations.")

print(f"--- Finished Experiment E13_SemAtt_DiscEm_NonComm ---")
```

E１４

```
# 14 --- 非可換＋断絶創発＋意味ラベル---
print("--- Executing Experiment E14_NonComm_DiscEm_SemAtt ---")
exp_to_run_name_e14 = "E14_NonComm_DiscEm_SemAtt"
```

```
config_to_run_e14 = next((config for config in experiment_configurations if config["name"] == exp_to_run_name_e14), None)

if config_to_run_e14:
    name, result = run_ablation_experiment(config_to_run_e14, global_variables_for_experiments)
    all_experiment_results_summary[name] = result
    print(f"\n--- Results for {name} ---")
    print(result)
else:
    print(f"Configuration for '{exp_to_run_name_e14}' not found.")
print(f"--- Finished Experiment E14_NonComm_DiscEm_SemAtt ---")
```

E１５

```
# E15 --- 断絶創発＋非可換＋意味ラベル
print("--- Executing Experiment E15_DiscEm_NonComm_SemAtt ---")

exp_to_run_name_e15 = "E15_DiscEm_NonComm_SemAtt"
config_to_run_e15 = next((config for config in experiment_configurations if config["name"] == exp_to_run_name_e15), None)

if config_to_run_e15:
    if 'all_experiment_results_summary' not in globals(): all_experiment_results_summary = {}
    if 'global_variables_for_experiments' not in globals(): raise NameError("global_variables_for_experiments is not defined.")

    name, result = run_ablation_experiment(config_to_run_e15, global_variables_for_experiments)
    all_experiment_results_summary[name] = result # 結果をグローバル辞書に格納
    print(f"\n--- Results for {name} ---")
    print(result)
else:
    print(f"Configuration for '{exp_to_run_name_e15}' not found in experiment_configurations.")

print(f"--- Finished Experiment E15_DiscEm_NonComm_SemAtt ---")
```

E１６

```
# --- セル (例: 16): 実験 E16_NonComm_SemAtt_Holo_DiscEm の実行 ---
print("--- Executing Experiment E16_NonComm_SemAtt_Holo_DiscEm ---")
exp_to_run_name_e16 = "E16_NonComm_SemAtt_Holo_DiscEm"
config_to_run_e16 = next((config for config in experiment_configurations if config["name"] == exp_to_run_name_e16), None)

if config_to_run_e16:
    if 'all_experiment_results_summary' not in globals(): all_experiment_results_summary = {}
    if 'global_variables_for_experiments' not in globals(): raise NameError("global_variables_for_experiments is not defined.")

    name, result = run_ablation_experiment(config_to_run_e16, global_variables_for_experiments)
    all_experiment_results_summary[name] = result
    print(f"\n--- Results for {name} ---")
    print(result)
else:
    print(f"Configuration for '{exp_to_run_name_e16}' not found.")
print(f"--- Finished Experiment E16_NonComm_SemAtt_Holo_DiscEm ---")
```

E１７

```
# --- セル (例: 17): 実験 E17_NonComm_Holo_SemAtt_Holo_DiscEm の実行 ---
print("--- Executing Experiment E17_NonComm_Holo_SemAtt_Holo_DiscEm ---")
```

```
exp_to_run_name_e17 = "E17_NonComm_Holo_SemAtt_Holo_DiscEm"
config_to_run_e17 = next((config for config in experiment_configurations if config["name"] == exp_to_run_name_e17), None)

if config_to_run_e17:
    # ... (run_ablation_experiment の呼び出しと結果格納は同様) ...
    name, result = run_ablation_experiment(config_to_run_e17, global_variables_for_experiments)
    all_experiment_results_summary[name] = result
    print(f"\n--- Results for {name} ---")
    print(result)
else:
    print(f"Configuration for '{exp_to_run_name_e17}' not found.")
print(f"--- Finished Experiment E17_NonComm_Holo_SemAtt_Holo_DiscEm ---")
```

E１８

```
# --- セル (例: 18): 実験 E18_Holo_NonComm_Holo_SemAtt_Holo_DiscEm の実行 ---
print("--- Executing Experiment E18_Holo_NonComm_Holo_SemAtt_Holo_DiscEm ---")
exp_to_run_name_e18 = "E18_Holo_NonComm_Holo_SemAtt_Holo_DiscEm"
config_to_run_e18 = next((config for config in experiment_configurations if config["name"] == exp_to_run_name_e18), None)

if config_to_run_e18:
    # ... (run_ablation_experiment の呼び出しと結果格納は同様) ...
    name, result = run_ablation_experiment(config_to_run_e18, global_variables_for_experiments)
    all_experiment_results_summary[name] = result
    print(f"\n--- Results for {name} ---")
    print(result)
else:
    print(f"Configuration for '{exp_to_run_name_e18}' not found.")
print(f"--- Finished Experiment E18_Holo_NonComm_Holo_SemAtt_Holo_DiscEm ---")
```

最終結果セル

```
# --- 最終結果セル: 全実験結果のサマリー表示と報告書出力 ---
import pandas as pd
import matplotlib.pyplot as plt
import datetime
import numpy as np # NaNを扱うため

print("--- Generating Final Experiment Report ---")

# all_experiment_results_summary と experiment_configurations が定義されているか確認
if 'all_experiment_results_summary' not in globals() or not isinstance(all_experiment_results_summary, dict):
    print("ERROR: 'all_experiment_results_summary' dictionary is not defined or not a dictionary.")
    print("Please ensure all experiment execution cells have run correctly and stored their results.")
    # このセル以降の処理を中断したい場合は raise NameError(...) も検討
    all_experiment_results_summary = {} # 空の辞書として処理を続ける（レポートは限定的になる）

if 'experiment_configurations' not in globals() or not isinstance(experiment_configurations, list):
    print("ERROR: 'experiment_configurations' list is not defined or not a list.")
    print("Please ensure Cell B (or the cell defining experiment_configurations) has been executed.")
    experiment_configurations = [] # 空のリストとして処理を続ける

if not all_experiment_results_summary:
    print("No experiment results found in 'all_experiment_results_summary' to generate a report.")
else:
    print(f"Found {len(all_experiment_results_summary)} entries in all_experiment_results_summary.")
    report_data = []
```

```
for config_name_key, metrics_dict_value in sorted(all_experiment_results_summary.items()):
    row = {'Experiment Name': config_name_key}

    config_details_found = False
    for config_detail in experiment_configurations:
        if config_detail.get('name') == config_name_key:
            current_tech_flags = config_detail.get('tech_flags', {})
            tech_flags_str_parts = []

            # module_order があればそれを元に、なければ tech_flags の True のものをリストアップ
            if "module_order" in current_tech_flags and isinstance(current_tech_flags["module_order"], list):
                for module_key_in_order in current_tech_flags["module_order"]:
                    # tech_flags の use_... が True かも確認
                    # モジュール名から "use_" と "_module" を除いたものを表示名とする
                    # 例: "semantic_attention_module" -> "semantic_attention"
                    display_name = module_key_in_order.replace("use_", "").replace("_module", "")

                    # 有効化フラグも確認 (より丁寧にするなら)
                    is_module_enabled_flag1 = current_tech_flags.get(f"use_{display_name}_module", False)
                    is_module_enabled_flag2 = current_tech_flags.get(f"use_{module_key_in_order}", False) # 完全一致も考慮
                    if is_module_enabled_flag1 or is_module_enabled_flag2:
                        tech_flags_str_parts.append(display_name)
            else: # module_order がない場合
                for flag, enabled in sorted(current_tech_flags.items()):
                    if enabled and flag.startswith("use_"):
                        tech_flags_str_parts.append(flag.replace("use_", "").replace("_module", ""))
            row['Enabled Modules'] = " -> ".join(tech_flags_str_parts) if tech_flags_str_parts else "Baseline"

            module_configs_str_parts = []
            for mc_key, mc_val in sorted(config_detail.get('module_configs', {}).items()):
                module_configs_str_parts.append(f"{mc_key.replace('_module','')} Params: {str(mc_val)}") # mc_valをstrに
            row['Module Parameters'] = "; ".join(module_configs_str_parts) if module_configs_str_parts else "N/A"
            config_details_found = True
            break
    if not config_details_found:
        row['Enabled Modules'] = "N/A"
        row['Module Parameters'] = "N/A"

    # metrics_dict_value が辞書であることを確認
    if isinstance(metrics_dict_value, dict):
        row.update({k: v for k, v in metrics_dict_value.items() if isinstance(v, (int, float, str, bool, np.number))})
    else:
        print(f"Warning: metrics_dict_value for {config_name_key} is not a dictionary: {metrics_dict_value}")
    report_data.append(row)

if not report_data:
    print("No data processed for the report DataFrame.")
else:
    report_df = pd.DataFrame(report_data)
    print("\n--- DataFrame for Report ---")
    if not report_df.empty:
        print(report_df) # DataFrameの内容を表示

        # --- 1. CSVファイルとして詳細データを出力 ---
        csv_filename = f"ablation_study_final_results_{datetime.datetime.now().strftime('%Y%m%d_%H%M%S')}.csv"
        try:
            report_df.to_csv(csv_filename, index=False, encoding='utf-8-sig')
            print(f"\nDetailed experiment results saved to {csv_filename}")
            # from google.colab import files # Colab環境でダウンロードする場合
            # files.download(csv_filename)
        except Exception as e:
```

```
    print(f"Error saving CSV: {e}")

# --- 2. テキストベースの報告書を作成 ---
report_text_parts_list = [] # リスト名を変更
report_text_parts_list.append("==========================================")
report_text_parts_list.append("     アブレーション実験結果報告書")
report_text_parts_list.append("==========================================")
report_text_parts_list.append(f"報告日時: {datetime.datetime.now().strftime('%Y年%m月%d日 %H時%M分%S秒')}")
report_text_parts_list.append("\n--- 実験概要 ---")
report_text_parts_list.append(f"実施した実験構成数: {len(experiment_configurations)}")
if 'BASE_MODEL_NAME' in globals():
    report_text_parts_list.append(f"ベースラインモデル: {BASE_MODEL_NAME}")

report_text_parts_list.append("\n--- 結果サマリーテーブル (DataFrame to_string) ---")
try:
    report_text_parts_list.append(report_df.to_string(index=False, float_format="%.4f", na_rep='N/A'))
except Exception as e:
    report_text_parts_list.append(f"Error formatting table with to_string: {e}")

report_text_parts_list.append("\n--- 主要メトリクス (eval_accuracy) の考察ポイント ---")
if 'eval_accuracy' not in report_df.columns or report_df['eval_accuracy'].isnull().all():
    report_text_parts_list.append("eval_accuracyに関する有効な結果がありません。")
else:
    try:
        # 'E0_Baseline' が存在するか確認
        if 'E0_Baseline' in report_df['Experiment Name'].values:
            baseline_accuracy_series = report_df[report_df['Experiment Name'] == 'E0_Baseline']['eval_accuracy']
            if not baseline_accuracy_series.empty and not pd.isna(baseline_accuracy_series.iloc[0]):
                baseline_accuracy = baseline_accuracy_series.iloc[0]
                report_text_parts_list.append(f"- ベースライン (E0_Baseline) の eval_accuracy: {baseline_accuracy:.4f}")

                for index, row in report_df.iterrows():
                    if row['Experiment Name'] != 'E0_Baseline' and pd.notna(row['eval_accuracy']):
                        change = row['eval_accuracy'] - baseline_accuracy
                        change_percent = (change / baseline_accuracy) * 100 if baseline_accuracy != 0 else float('inf')
                        report_text_parts_list.append(
                            f"- {row['Experiment Name']}: eval_accuracy = {row['eval_accuracy']:.4f} "
                            f"(ベースライン比: {change:+.4f}, {change_percent:+.2f}%)"
                        )
                    elif row['Experiment Name'] != 'E0_Baseline':
                        report_text_parts_list.append(f"- {row['Experiment Name']}: eval_accuracy = N/A")
            else:
                report_text_parts_list.append("ベースラインのeval_accuracyが見つからないか、N/Aです。")
        else:
            report_text_parts_list.append("E0_Baseline の結果が見つかりません。")
    except Exception as e: # より広範なエラーキャッチ
        report_text_parts_list.append(f"eval_accuracy考察中にエラー: {e}")

report_text_parts_list.append("\n--- 定性的な考察（例） ---")
report_text_parts_list.append("（ここに、各モジュールの効果や、特定の入力に対する挙動の変化など、数値だけでは見えない考察を記述します）")
report_text_parts_list.append("\n==========================================")
report_text_parts_list.append("            報告書終わり")
report_text_parts_list.append("==========================================")

final_report_text_content = "\n".join(report_text_parts_list) # 変数名を変更
print(f"\n{final_report_text_content}")

report_filename_txt = f"ablation_study_summary_report_{datetime.datetime.now().strftime('%Y%m%d_%H%M%S')}.txt" # 変数名を変更
try:
```

```
            with open(report_filename_txt, "w", encoding="utf-8") as f: # 変数名を変更
                f.write(final_report_text_content)
            print(f"\nSummary report saved to {report_filename_txt}")
        except Exception as e:
            print(f"Error saving text report: {e}")

        # --- 3. 主要メトリクスのグラフ化 (eval_accuracyの棒グラフ) ---
        print("\n--- Generating Accuracy Comparison Graph ---")
        if 'eval_accuracy' not in report_df.columns or report_df['eval_accuracy'].isnull().all():
            print("No valid eval_accuracy data to plot for the graph.")
        else:
            try:
                # NaNをプロット前に0に置換 (あるいはNaNを除外する処理も検討可)
                accuracies_for_plot_graph = pd.to_numeric(report_df['eval_accuracy'], errors='coerce').fillna(0.0) # 変数名を変更
                config_names_for_plot_graph = report_df['Experiment Name'] # 変数名を変更

                plt.figure(figsize=(12, max(6, len(report_df) * 0.5)))
                bars = plt.barh(config_names_for_plot_graph, accuracies_for_plot_graph, color='lightcoral') # 色を変更
                plt.xlabel("Evaluation Accuracy (eval_accuracy)") # X軸ラベルを明確化
                plt.ylabel("Experiment Configuration")
                plt.title("Ablation Study: eval_accuracy Comparison") # タイトルを明確化
                plt.gca().invert_yaxis()
                plt.xticks(rotation=30, ha="right")

                for bar in bars:
                    width = bar.get_width()
                    plt.text(width + 0.005, bar.get_y() + bar.get_height()/2,
                             f'{width:.4f}', va='center', ha='left')

                plt.tight_layout()
                graph_filename_png =
f"eval_accuracy_comparison_{datetime.datetime.now().strftime('%Y%m%d_%H%M%S')}.png" # 変数名を変更
                plt.savefig(graph_filename_png) # 変数名を変更
                print(f"Accuracy comparison graph saved to {graph_filename_png}")
                plt.show()
            except Exception as e:
                print(f"Error generating graph: {e}")
                import traceback
                traceback.print_exc() # エラー詳細を表示
    else:
        print("Report DataFrame is empty. Cannot generate CSV, text report, or graph.")

print("\n--- Report Generation Cell Complete ---")
```