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

エコツーラボ合同会社
猪澤也寸志
polyp@ webman.jp
（2025年5月27日現地時間JST）

セル1

```
# 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2

```
# 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3

```
# --- 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４

```
# --- 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 = 3                  # 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 5

```
# --- 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 = 100  # 例: 10,000件 (または len(final_tokenized_train) で全件)
    NUM_EVAL_SAMPLES_FULL = 25    # 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 6

```
# --- 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 7

```
# --- 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 8

```
# --- 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 9

```
# --- セルA: 統合モデルとDebugTrainerの定義 ---
from transformers import PreTrainedModel, AutoModel, AutoConfig, Trainer # Trainerもインポート
from transformers.modeling_outputs import SequenceClassifierOutput
from torch.nn import CrossEntropyLoss
# (SemanticConceptModule, DirectNonCommutativeModule, DiscontinuityEmergenceModule は
#  それぞれセル5, 6, 7で定義済みであることを前提とします)

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

        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
        self.tech_modules = nn.ModuleDict() # 統一してModuleDictを使用
        self.tech_flags = tech_flags if tech_flags is not None else {}
        self.module_configs = module_configs if module_configs is not None else {}

        self.module_order = self.tech_flags.get("module_order", []) # モジュール適用順序をtech_flagsから取得

        # モジュールをmodule_orderに従って初期化
        for module_name_key in self.module_order:
            if module_name_key == "semantic_attention_module" and self.tech_flags.get("use_semantic_attention_module", False):
                scm_config = self.module_configs.get(module_name_key, {})
                module_output_dim = scm_config.get("output_dim", current_dim)
                self.tech_modules[module_name_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
                )
                current_dim = module_output_dim
                print(f"{module_name_key} enabled and initialized.")
            elif module_name_key == "direct_non_commutative_module" and
self.tech_flags.get("use_direct_non_commutative_module", False):
                dnc_config = self.module_configs.get(module_name_key, {})
                module_output_dim = dnc_config.get("output_dim", current_dim)
                self.tech_modules[module_name_key] = DirectNonCommutativeModule(
                    input_dim=current_dim, output_dim=module_output_dim,
                    dropout=dnc_config.get("dropout", 0.2), module_specific_config=dnc_config
                )
                current_dim = module_output_dim
                print(f"{module_name_key} enabled and initialized.")
            elif module_name_key == "discontinuity_emergence_module" and
self.tech_flags.get("use_discontinuity_emergence_module", False):
                dem_config = self.module_configs.get(module_name_key, {})
                module_output_dim = dem_config.get("output_dim", current_dim)
                self.tech_modules[module_name_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), module_specific_config=dem_config
                )
                current_dim = module_output_dim
                print(f"{module_name_key} enabled and initialized.")
```

```
            # 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)
        print(f"EnhancedTransformerForSequenceClassification initialized. Classifier input dim: {current_dim}. Applied modules (in order): {self.module_order}")

    def forward(
        self, input_ids=None, attention_mask=None, token_type_ids=None, labels=None,
        return_dict=None, **kwargs ):
        # print(f"--- EnhancedTransformer.forward CALLED (is_training: {self.training}) ---")

        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_hidden_states = transformer_outputs.last_hidden_state

        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.module_order:
            if module_name_key in self.tech_modules: # 有効化フラグは__init__でチェック済みなので、ここでは存在確認のみ
                # print(f"  Applying module: {module_name_key}")
                module_output_dict = self.tech_modules[module_name_key](current_hidden_states, attention_mask=attention_mask)
                current_hidden_states = 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 # キー名が重複しないように

        pooled_output = current_hidden_states[:, 0]
        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))
            # print(f"  Forward CALCULATED loss: {loss.item() if loss is not None else 'None'}")

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

        final_outputs_tuple = (logits,)
        # ベースモデルの出力とモジュールの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,) # ベースの全層のhidden_states
        if hasattr(transformer_outputs, 'attentions') and transformer_outputs.attentions is not None:
            final_outputs_tuple += (transformer_outputs.attentions,) # ベースの全層の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

class DebugTrainer(Trainer): # (変更なし、以前の定義のまま)
    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 (multi-module ready) and DebugTrainer defined.")
```

セル10

```
# --- セルB: 実験構成リストと共通実験実行関数の定義 ---

# (セル1〜Aで、必要なグローバル変数やクラスが定義済みであることを前提とします)
# (特に、BASE_MODEL_NAME, NUM_LABELS, OUTPUT_DIR_BASE, LEARNING_RATE, BATCH_SIZE, NUM_EPOCHS,
#  tokenizer, hf_model_config, train_dataset, eval_dataset, compute_metrics,
#  EnhancedTransformerForSequenceClassification, SemanticConceptModule,
#  DirectNonCommutativeModule, DiscontinuityEmergenceModule, DebugTrainer, Dataset など)

print("--- Cell B: Defining Experiment Configurations, Global Vars Pack, and Execution Function ---")

# --- 1. 前提となるグローバル変数の確認 (念のため) ---
required_globals_cell_B = [
    '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'
]
missing_globals_check_B = False
for var_name in required_globals_cell_B:
    if var_name not in globals():
        print(f"ERROR in Cell B: Global variable '{var_name}' is not defined!")
        missing_globals_check_B = True
if missing_globals_check_B:
    raise NameError("One or more required global variables for Cell B are not defined. Please check prerequisite cells.")
else:
    print("All prerequisite global variables for Cell B seem to be defined.")

# --- 2. 実験構成の定義 (お客様の全パターンを網羅) ---
# モジュール名は EnhancedTransformerForSequenceClassification の __init__ で使ったキー名と一致させる
# module_order リストでモジュールの適用順序を指定
experiment_configurations = [
    # --- ベースライン ---
    {"name": "E0_Baseline", "tech_flags": {"module_order": []}, "module_configs": {}},
    # --- 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}}},
    {"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}}},
    {"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}}},

    # --- 2モジュールの組み合わせ (順序考慮) ---
    # (セル11相当) 1+意味ラベル→非可換
    {"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}}},
    # (セル12相当) 1+非可換→意味ラベル
    {"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}}},

    # --- 3モジュールの組み合わせ (お客様のリストアップに基づき) ---
    # (セル13相当) 1+意味ラベル→非可換→断絶創発
    {"name": "E6_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}}},
    # (セル14相当) 1+非可換→意味ラベル→断絶創発
    {"name": "E7_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": { # モジュール設定はE6と同じと仮定 (必要なら変更)
         "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}}},
    # (セル15相当) 1+断絶創発→意味ラベル→非可換
    {"name": "E8_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": { # モジュール設定はE6と同じと仮定 (必要なら変更)
         "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}}},
    # 他に必要な組み合わせや順序があれば、同様の形式で追加してください。
]
```

```
print(f"Total experiment configurations defined: {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
}
print("Global variables for experiments packaged.")

# --- 4. 共通実験実行関数 ---
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}")

    # グローバル変数を辞書から取得
    hf_cfg = global_vars_dict['hf_model_config']
    base_model = global_vars_dict['BASE_MODEL_NAME']
    n_labels = global_vars_dict['NUM_LABELS']
    output_dir_base = 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 = 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 = global_vars_dict['DebugTrainer']

    # eval_dataset のカラム名再確認 (デバッグ用)
    if ev_dataset is not None and isinstance(ev_dataset, global_vars_dict['Dataset']): # Datasetクラスも辞書から取得
        print(f"  Eval dataset column names for this experiment ({config_name}): {ev_dataset.column_names}")
        if 'labels' not in ev_dataset.column_names:
            print(f"    WARNING for {config_name}: 'labels' column not found in eval_dataset! This WILL cause evaluation issues.")
    else:
        print(f"  Warning for {config_name}: eval_dataset not found or not a Dataset object.")

    model = enh_transformer_class(
        hf_config=hf_cfg,
        base_model_name=base_model,
        num_labels=n_labels,
        tech_flags=tech_flags,
        module_configs=module_cfgs
```

```
    )

    current_output_dir = f"{output_dir_base}{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, # ディスク節約のため
    )

    trainer = dbg_trainer_class(
        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)}

    return config_name, eval_results_dict

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

print("Cell B: All experiment configurations and the execution function run_ablation_experiment are defined.")
```

セル11

```
# (セルBの最後、またはセル8の冒頭に配置)
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, # または Trainer
    'Dataset': Dataset # Datasetクラスも渡しておく (run_ablation_experiment内で使う場合)
}
print("Global variables for experiments packaged.")
```

セル12

```
# --- セル8: 実験 E0_Baseline の実行 ---
print("--- Executing Experiment E0_Baseline ---")

exp_to_run_name_e0 = "E0_Baseline"
config_to_run_e0 = next((config for config in experiment_configurations if config["name"] == exp_to_run_name_e0), None)

if config_to_run_e0:
    if 'all_experiment_results_summary' not in globals(): all_experiment_results_summary = {} #念のため初期化
    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_e0, 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_e0}' not found in experiment_configurations.")

print(f"--- Finished Experiment E0_Baseline ---")
```

セル13

```
# --- セル9: 実験 E1_SemanticAttention の実行 ---
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 ---")
```

セル14

```
# --- セル10: 実験 E2_DirectNonCommutative の実行 ---
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 ---")
```

セル15

```
# --- セル11: 実験 E3_DiscontinuityEmergence の実行 ---
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 ---")
```

セル16

```
# --- セル12: 実験 E5_NonComm_then_SemAtt の実行 ---
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 ---")
```

セル17

```
# --- セル13: 実験 E6_SemAtt_NonComm_DiscEm の実行 ---
print("--- Executing Experiment E6_SemAtt_NonComm_DiscEm ---")
exp_to_run_name_e6 = "E6_SemAtt_NonComm_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:
    # ... (run_ablation_experiment の呼び出しと結果格納は同様) ...
    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 E6_SemAtt_NonComm_DiscEm ---")
```

セル18

```
# --- セル14: 実験 E7_NonComm_SemAtt_DiscEm の実行 ---
print("--- Executing Experiment E7_NonComm_SemAtt_DiscEm ---")

exp_to_run_name_e7 = "E7_NonComm_SemAtt_DiscEm"
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_NonComm_SemAtt_DiscEm ---")
```

セル19

```
# --- セル15: 実験 E8_DiscEm_SemAtt_NonComm の実行 ---
print("--- Executing Experiment E8_DiscEm_SemAtt_NonComm ---")

exp_to_run_name_e8 = "E8_DiscEm_SemAtt_NonComm"
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 E8_DiscEm_SemAtt_NonComm ---")
```

セル20

```
# --- セル12: 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

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

if 'all_experiment_results_summary' not in globals() or not all_experiment_results_summary:
    print("No experiment results found to summarize.")
else:
    report_data = []
    for config_name_key, metrics_dict_value in sorted(all_experiment_results_summary.items()):
        row = {'Experiment Name': config_name_key} # 列名を変更

        config_details_found = False
        for config_detail in experiment_configurations: # セルBで定義したリストを参照
            if config_detail['name'] == config_name_key:
                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 config_detail.get('tech_flags', {}):
                    for module_key_in_order in config_detail['tech_flags']['module_order']:
                        # tech_flags の use_... が True かも確認した方がより丁寧
                        if config_detail['tech_flags'].get(f"use_{module_key_in_order.replace('_module','').lower()}_module", False) or \
                           config_detail['tech_flags'].get(f"use_{module_key_in_order}", False): # 完全一致も考慮
                           tech_flags_str_parts.append(module_key_in_order.replace("use_", "").replace("_module", ""))
                else: # module_order がない場合 (古い形式など)
                    for flag, enabled in sorted(config_detail.get('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: {mc_val}")
                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"

        row.update({k: v for k, v in metrics_dict_value.items() if isinstance(v, (int, float, str, bool))})
        report_data.append(row)

    report_df = pd.DataFrame(report_data)
    print("\n--- DataFrame for Report ---")
    print(report_df)

    # CSV出力 (以前のコードと同様)
    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
        # files.download(csv_filename)
    except Exception as e:
```

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

    # テキストレポート出力 (以前のコードと同様、report_df を使用)
    # ... (省略、必要なら以前のコードをここに挿入) ...

    # グラフ出力 (以前のコードと同様、report_df を使用)
    print("\n--- Generating Accuracy Comparison Graph ---")
    if 'Accuracy' not in report_df.columns or report_df['Accuracy'].isnull().all():
        print("No valid accuracy data to plot.")
    else:
        try:
            accuracies_for_plot = pd.to_numeric(report_df['Accuracy'], errors='coerce').fillna(0)
            config_names_for_plot = report_df['Experiment Name'] # DataFrameの列名に合わせる

            plt.figure(figsize=(12, max(6, len(report_df) * 0.6)))
            bars = plt.barh(config_names_for_plot, accuracies_for_plot, color='lightgreen')
            plt.xlabel("Evaluation Accuracy")
            plt.ylabel("Experiment Configuration")
            plt.title("Ablation Study: Accuracy Comparison")
            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 = f"accuracy_comparison_final_{datetime.datetime.now().strftime('%Y%m%d_%H%M%S')}.png"
            plt.savefig(graph_filename)
            print(f"Accuracy comparison graph saved to {graph_filename}")
            plt.show()
            # from google.colab import files
            # files.download(graph_filename)
        except Exception as e:
            print(f"Error generating graph: {e}")

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

実験結果

```
Final Experiment Report ---

--- DataFrame for Report ---
                Experiment Name  \
0                   E0_Baseline
1          E1_SemanticAttention
2       E2_DirectNonCommutative
3    E3_DiscontinuityEmergence
4       E5_NonComm_then_SemAtt
5     E6_SemAtt_NonComm_DiscEm
6     E7_NonComm_SemAtt_DiscEm
7     E8_DiscEm_SemAtt_NonComm
```

```
                                      Enabled Modules  \
0                                                Baseline
1                                    semantic_attention
2                                 direct_non_commutative
3                                discontinuity_emergence
4       direct_non_commutative -> semantic_attention
5  semantic_attention -> direct_non_commutative -...
6  direct_non_commutative -> semantic_attention -...
7  discontinuity_emergence -> semantic_attention ...

                                      Module Parameters
eval_accuracy  \
0                                                   N/A
0.88
1  semantic_attention Params: {'num_concepts': 32...
0.60
2  direct_non_commutative Params: {'output_dim': ...
0.60
3  discontinuity_emergence Params: {'output_dim':...
0.60
4  direct_non_commutative Params: {'output_dim': ...
0.68
5  direct_non_commutative Params: {'output_dim': ...
0.84
6  direct_non_commutative Params: {'output_dim': ...
0.76
7  direct_non_commutative Params: {'output_dim': ...
0.76

   eval_loss  eval_runtime  eval_samples_per_second
eval_steps_per_second  \
0   0.410606        0.5728                   43.643
43.643
1   1.565224        0.5758                   43.417
43.417
2   1.605075        2.9909                    8.359
8.359
3   1.089734        0.5314                   47.042
47.042
4   1.145913        3.4214                    7.307
7.307
```

```
5   0.474421        3.1970                      7.820
7.820
6   1.000389        3.7171                      6.726
6.726
7   0.811572        3.0007                      8.331
8.331
```

```
   epoch
0    3.0
1    3.0
2    3.0
3    3.0
4    3.0
5    3.0
6    3.0
7    3.0
```

```
Detailed experiment results saved to
ablation_study_final_results_20250527_091328.csv

--- Generating Accuracy Comparison Graph ---
No valid accuracy data to plot.
--- Report Generation Cell Complete ---
```