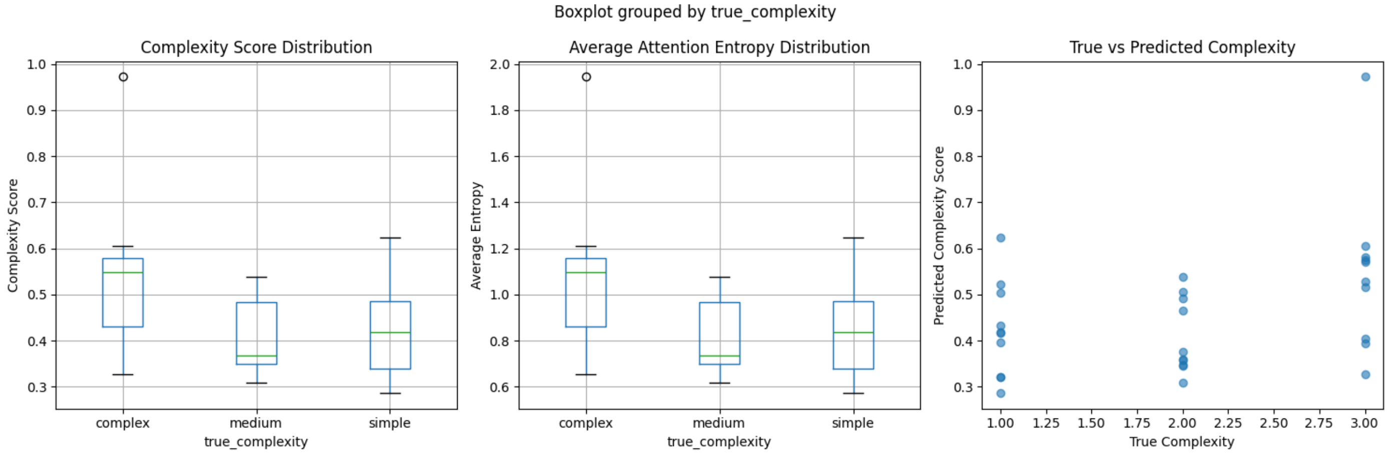
```
"""
Optimized Stage 1 Experiment: Combining comprehensive statistical analysis with routing decisions
"""
import torch
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
from transformers import AutoTokenizer, AutoModelForCausalLM
from typing import Dict, List


class OptimizedAttentionEntropyAnalyzer:
    def __init__(self, entropy_threshold=2.0):
        self.entropy_threshold = entropy_threshold
        self.complexity_history = []

    def calculate_attention_entropy(self, attention_weights):
        """
        Calculate attention entropy - supports multiple computation methods
        Input: attention_weights [num_heads, seq_len] or [seq_len, seq_len]
        """
        if len(attention_weights.shape) == 3:  # [num_heads, seq_len, seq_len]
            # For full attention matrix, calculate entropy for each query
            entropies = []
            for head in attention_weights:
                head_entropies = []
                for i in range(head.shape[0]):
                    attn_dist = head[i] + 1e-9  # Avoid log(0)
                    entropy = -torch.sum(attn_dist * torch.log(attn_dist))
                    head_entropies.append(entropy.item())
                entropies.append(np.mean(head_entropies))
            return entropies

        else:  # [num_heads, seq_len] - attention distribution for single token
            entropies = []
            for head_attn in attention_weights:
                head_attn = head_attn + 1e-9
                entropy = -torch.sum(head_attn * torch.log(head_attn))
                entropies.append(entropy.item())
            return entropies

    def predict_complexity(self, attention_weights):
        """Predict complexity based on attention entropy"""
        entropies = self.calculate_attention_entropy(attention_weights)
        avg_entropy = np.mean(entropies)
        max_entropy = np.max(entropies)
        entropy_std = np.std(entropies)

        # Multi-dimensional complexity evaluation
        complexity_score = min(avg_entropy / self.entropy_threshold, 1.0)

        return {
            'complexity_score': complexity_score,
            'avg_entropy': avg_entropy,
            'max_entropy': max_entropy,
            'entropy_std': entropy_std,
            'head_entropies': entropies,
            'is_complex': complexity_score > 0.5
        }

    def should_route_to_cloud(self, attention_weights, threshold=0.5):
        """Routing decision"""
        result = self.predict_complexity(attention_weights)
        return result['complexity_score'] > threshold, result


class ComprehensiveBaselineExperiment:
    def __init__(self, model_name="microsoft/DialoGPT-small"):
        self.tokenizer = AutoTokenizer.from_pretrained(model_name)
        self.model = AutoModelForCausalLM.from_pretrained(model_name, output_attentions=True)
        self.analyzer = OptimizedAttentionEntropyAnalyzer()

        # Set pad_token
        if self.tokenizer.pad_token is None:
            self.tokenizer.pad_token = self.tokenizer.eos_token
        # Extended test dataset
        self.task_dataset = {
            "simple": [
                "What is 2+2?",
                "The capital of France is",
                "My name is",
                "What color is the sky?",
                "How many days in a week?",
                "What is water made of?",
                "The sun rises in the",
                "1+1 equals",
                "Cats are",
                "The alphabet starts with"
            ],
            "medium": [
                "Why do objects fall down?",
                "How does a bicycle work?",
                "What causes rain?",
                "Why is the ocean salty?",
                "How do plants make food?",
                "What makes ice float?",
                "Why do we have seasons?",
                "How do computers work?",
                "What is electricity?",
                "Why do we dream?"
            ],
            "complex": [
                "Explain the relationship between quantum mechanics and general relativity",
                "Analyze the economic impact of artificial intelligence on employment",
                "What would happen if gravity suddenly became twice as strong?",
                "Discuss the ethical implications of genetic engineering",
                "How might climate change affect global food security?",
                "Evaluate the societal effects of social media on democracy",
                "If I have 3 apples and give away 1.5 apples, then buy 2.7 more apples, what's the philosophical meaning?",
                "Compare the advantages and disadvantages of different renewable energy sources",
                "How do cultural differences affect international business negotiations?",
                "What are the long-term consequences of space exploration for humanity?"
            ]
        }

    def extract_attention_features(self, text, method="last_token"):
        """Extract attention features"""
        inputs = self.tokenizer(text, return_tensors="pt", padding=True, truncation=True, max_length=512)

        with torch.no_grad():
            outputs = self.model(**inputs, output_attentions=True)

        # Get last layer attention
        last_attention = outputs.attentions[-1][0]  # [num_heads, seq_len, seq_len]

        if method == "last_token":
            # Analyze attention pattern of the last valid token
            seq_len = inputs['attention_mask'].sum().item()
            last_token_attn = last_attention[:, seq_len-1, :]  # [num_heads, seq_len]
            return last_token_attn

        elif method == "average":
            # Average attention across all tokens
            return last_attention

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

    def run_single_task(self, text, complexity_label):
        """Run single task"""
        attention_weights = self.extract_attention_features(text)
        result = self.analyzer.predict_complexity(attention_weights)

        # Add true label
        result['true_complexity'] = complexity_label
        result['task'] = text

        return result

    def run_comprehensive_experiment(self):
        """Run comprehensive experiment"""
        print("🧪 Running optimized Stage 1 experiment...")
        print("🧪 Running optimized Stage 1 experiment...")

        all_results = []

        # Run all tasks
        for complexity, tasks in self.task_dataset.items():
            print(f"\n📊 Processing {complexity} tasks...")

            for task in tasks:
                result = self.run_single_task(task, complexity)
                all_results.append(result)
                print(f"'{task[:50]}...' -> complexity={result['complexity_score']:.3f}")

        # Convert to DataFrame for analysis
        df = pd.DataFrame(all_results)

        return self.analyze_results(df)

    def analyze_results(self, df):
        """Comprehensive result analysis"""
        print("\n" + "="*60)
        print("📈 EXPERIMENTAL RESULTS ANALYSIS")
        print("="*60)

        # 1. Descriptive statistics
        summary = df.groupby('true_complexity')['complexity_score'].agg([
            'count', 'mean', 'std', 'min', 'max'
        ]).round(3)
        print("\n1. Descriptive Statistics:")
        print(summary)

        # 2. Visualization
        self.plot_results(df)

        # 3. Statistical tests
        self.statistical_tests(df)

        # 4. Correlation analysis
        self.correlation_analysis(df)

        # 5. Routing decision analysis
        self.routing_analysis(df)

        return df

    def plot_results(self, df):
        """Result visualization"""
        plt.figure(figsize=(15, 5))

        # Box plot for complexity scores
        plt.subplot(1, 3, 1)
        df.boxplot(column='complexity_score', by='true_complexity', ax=plt.gca())
        plt.title('Complexity Score Distribution')
        plt.ylabel('Complexity Score')

        # Box plot for average entropy
        plt.subplot(1, 3, 2)
        df.boxplot(column='avg_entropy', by='true_complexity', ax=plt.gca())
        plt.title('Average Attention Entropy Distribution')
        plt.ylabel('Average Entropy')

        # Scatter plot
        plt.subplot(1, 3, 3)
        complexity_mapping = {'simple': 1, 'medium': 2, 'complex': 3}
        df['complexity_numeric'] = df['true_complexity'].map(complexity_mapping)
        plt.scatter(df['complexity_numeric'], df['complexity_score'], alpha=0.6)
        plt.xlabel('True Complexity')
        plt.ylabel('Predicted Complexity Score')
        plt.title('True vs Predicted Complexity')

        plt.tight_layout()
        plt.show()

    def statistical_tests(self, df):
        """Statistical significance testing"""
        print("\n2. Statistical Significance Tests:")

        simple_scores = df[df['true_complexity'] == 'simple']['complexity_score']
        medium_scores = df[df['true_complexity'] == 'medium']['complexity_score']
        complex_scores = df[df['true_complexity'] == 'complex']['complexity_score']

        # ANOVA test
        f_stat, p_value = stats.f_oneway(simple_scores, medium_scores, complex_scores)
        print(f"ANOVA F-statistic: {f_stat:.4f}, p-value: {p_value:.4f}")

        if p_value < 0.05:
            print("✅ Significant difference between groups (p < 0.05)")
        else:
            print("❌ No significant difference between groups (p >= 0.05)")

        # Pairwise comparison
        from scipy.stats import ttest_ind
        t1, p1 = ttest_ind(simple_scores, complex_scores)
        print(f"Simple vs Complex tasks t-test: t={t1:.3f}, p={p1:.4f}")

    def correlation_analysis(self, df):
        """Correlation analysis"""
        print("\n3. Correlation Analysis:")

        complexity_mapping = {'simple': 1, 'medium': 2, 'complex': 3}
        df['complexity_numeric'] = df['true_complexity'].map(complexity_mapping)

        correlation = df['complexity_score'].corr(df['complexity_numeric'])
        print(f"Correlation between complexity score and true complexity: {correlation:.4f}")

        if correlation > 0.5:
            print("✅ Strong positive correlation - Hypothesis validated!")
        elif correlation > 0.3:
            print("⚠️ Moderate correlation - Some effectiveness but needs improvement")
        else:
            print("❌ Weak correlation - Need to reconsider methodology")

    def routing_analysis(self, df):
        """Routing decision analysis"""
        print("\n4. Routing Decision Analysis:")

        # Calculate routing decisions for each complexity level
        routing_stats = df.groupby('true_complexity')['is_complex'].agg([
            'count', 'sum', lambda x: (x.sum() / len(x) * 100)
        ]).round(1)
        routing_stats.columns = ['Total', 'Routed to Cloud', 'Routing Rate (%)']
        print(routing_stats)

        # Ideal case: simple tasks not routed, complex tasks routed
        simple_correct = (df[df['true_complexity'] == 'simple']['is_complex'] == False).sum()
        complex_correct = (df[df['true_complexity'] == 'complex']['is_complex'] == True).sum()

        simple_total = len(df[df['true_complexity'] == 'simple'])
        complex_total = len(df[df['true_complexity'] == 'complex'])

        print(f"\nRouting Accuracy:")
        print(f"Simple task correct routing rate: {simple_correct/simple_total*100:.1f}%")
        print(f"Complex task correct routing rate: {complex_correct/complex_total*100:.1f}%")

    def save_results(self, df, filename="stage1_results.csv"):
        """Save results"""
        df.to_csv(filename, index=False)
        print(f"\n💾 Results saved to {filename}")


if __name__ == "__main__":
    # Run experiment
    experiment = ComprehensiveBaselineExperiment()
    results_df = experiment.run_comprehensive_experiment()

    # Save results
    experiment.save_results(results_df)

    print("\n🎉 Stage 1 experiment completed!")
```

```
The following generation flags are not valid and may be ignored: ['output_attentions']. Set `TRANSFORMERS_VERBOSITY=info` fo
The following generation flags are not valid and may be ignored: ['output_attentions']. Set `TRANSFORMERS_VERBOSITY=info` fo
🧪 Running optimized Stage 1 experiment...

📊 Processing simple tasks...
'What is 2+2?...' -> complexity=0.623
'The capital of France is...' -> complexity=0.396
'My name is...' -> complexity=0.287
'What color is the sky?...' -> complexity=0.504
'How many days in a week?...' -> complexity=0.522
'What is water made of?...' -> complexity=0.321
'The sun rises in the...' -> complexity=0.418
'1+1 equals...' -> complexity=0.432
'Cats are...' -> complexity=0.321
'The alphabet starts with...' -> complexity=0.417

📊 Processing medium tasks...
'Why do objects fall down?...' -> complexity=0.538
'How does a bicycle work?...' -> complexity=0.376
'What causes rain?...' -> complexity=0.464
'Why is the ocean salty?...' -> complexity=0.357
'How do plants make food?...' -> complexity=0.507
'What makes ice float?...' -> complexity=0.491
'Why do we have seasons?...' -> complexity=0.308
'How do computers work?...' -> complexity=0.345
'What is electricity?...' -> complexity=0.361
'Why do we dream?...' -> complexity=0.348

📊 Processing complex tasks...
'Explain the relationship between quantum mechanics...' -> complexity=0.528
'Analyze the economic impact of artificial intellig...' -> complexity=0.403
'What would happen if gravity suddenly became twice...' -> complexity=0.516
'Discuss the ethical implications of genetic engine...' -> complexity=0.327
'How might climate change affect global food securi...' -> complexity=0.581
'Evaluate the societal effects of social media on d...' -> complexity=0.394
'If I have 3 apples and give away 1.5 apples, then ...' -> complexity=0.972
'Compare the advantages and disadvantages of differ...' -> complexity=0.605
'How do cultural differences affect international b...' -> complexity=0.574
'What are the long-term consequences of space explo...' -> complexity=0.570

============================================================
📈 EXPERIMENTAL RESULTS ANALYSIS
============================================================

1. Descriptive Statistics:
                 count   mean    std    min    max
true_complexity
complex             10  0.547  0.176  0.327  0.972
medium              10  0.409  0.082  0.308  0.538
simple              10  0.424  0.104  0.287  0.623
```



```
2. Statistical Significance Tests:
ANOVA F-statistic: 3.5324, p-value: 0.0434
✅ Significant difference between groups (p < 0.05)
Simple vs Complex tasks t-test: t=-1.900, p=0.0735

3. Correlation Analysis:
Correlation between complexity score and true complexity: 0.3704
⚠️ Moderate correlation - Some effectiveness but needs improvement

4. Routing Decision Analysis:
                 Total  Routed to Cloud  Routing Rate (%)
true_complexity
complex             10                7              70.0
medium              10                2              20.0
```