Benchmark de Estratégias de KV Cache - PETTEC

November 25, 2025

1 Tutorial de Benchmark de Estratégias de KV Cache - PETTEC

Este tutorial apresenta, passo a passo, a análise comparativa de estratégias para gerenciamento de KV cache em modelos de linguagem, relealizado pelos alunos do PETTEC e apresentado no Simpósio Unifei 2025.

Vamos medir tempo de geração, uso de memória, throughput (tokens por segundo) e taxa de sucesso das respostas para três estratégias: - **Sem cache** - **Dynamic cache** - **Static cache**

Ao final, obtenha gráficos e conclusões sobre desempenho prático de cada abordagem.

1.1 O que é KV Cache? Por que usar cache?

Modelos de linguagem como o Llama utilizam atenção causal, isto é, a cada novo token gerado, recalculam partes do histórico da conversa/conteúdo. O mecanismo de **KV cache** (chave/valor cache) permite guardar esses estados intermediários para evitar cálculos repetidos.

1.1.1 Vantagens do KV Cache

- Menor tempo de resposta na geração de textos longos ou interativos (conversação);
- Redução de uso de memória, pois reaproveita cálculos prévios;
- Essencial para aplicações em tempo real (chats, assistentes etc).

1.2 Técnicas comparadas neste tutorial

1. Sem Cache: - O modelo não armazena nenhum cálculo intermediário. Maior custo computacional, pois a cada geração tudo é recalculado desde o início. - Útil apenas em tarefas simples/pequenas, geralmente ineficiente para workflows reais.

2. Dynamic Cache: - Implementa cache de maneira flexível, alocando memória conforme cresce o histórico da conversa. - Equilíbrio entre uso de memória e velocidade. Padrão nos principais frameworks atuais.

3. Static Cache: - Pré-aloca uma área de memória fixa para o cache, tornando a computação de tokens extremamente rápida. - Excelente velocidade para contextos de tamanho previsível, menos flexível para históricos muito longos ou variantes de entrada.

Essas opções mostram o trade-off entre simplicidade, performance e uso de recursos, explicando a importância dos benchmarks em cenários reais de uso!

1.3 1. Importações e Configurações Iniciais

Vamos importar as bibliotecas necessárias e configurar o ambiente para executar o benchmark.

[1]:
```
# Importação das bibliotecas utilizadas
import torch
import numpy as np
import matplotlib.pyplot as plt
import time
from transformers import AutoTokenizer, AutoModelForCausalLM
from typing import List, Dict
```

1.4 2. Definição da Classe de Benchmark

A seguir está a classe que permite rodar os testes de comparação entre diferentes estratégias de cache. Ela possui métodos para medir tempo de geração, uso de memória, throughput e registrar as respostas para avaliação da taxa de sucesso. Execute esta célula para disponibilizar a classe.

1.4.1 Como usar as técnicas de KV Cache?

O uso do KV cache em modelos HuggingFace pode ser controlado por argumentos passados ao método de geração, especialmente: - **use_cache** (bool): ativa/desativa o cache (desativado = recalcula tudo do zero). - **cache_implementation** (str): pode ser 'dynamic' ou 'static' (para frameworks modernos e modelos compatíveis). - Outros parâmetros podem aparecer conforme o modelo/framework (veja sempre a documentação específica do modelo).

Exemplo básico:

```
outputs = model.generate(
    **inputs,
    use_cache=True,  # ativa o uso de cache
    cache_implementation="dynamic",  # ou "static" para cache estático
    max_new_tokens=100
)
```

Esses parâmetros são especialmente relevantes quando se deseja benchmarks, inferência rápida em pipelines de chatbots ou redução de custo computacional.

1.4.2 Documentação oficial e leitura recomendada

- Transformers: Generation with past key values (Use cache)
- Explicação sobre `use_cache` (pytorch docs)

Fique atento, pois nem todo modelo/função de geração expõe igualmente todos argumentos. Consulte sempre a documentação do modelo específico!

[2]:
```
import psutil, gc, warnings, os, csv
from typing import List, Dict, Any, Tuple
warnings.filterwarnings("ignore")

class KVCacheBenchmark:
```

```
  def __init__(self, model, tokenizer, device, model_name: str = "meta-llama/
↪Llama-3.2-1B"):
      self.model = model
      self.tokenizer = tokenizer
      self.device = device
      self.model_name = model_name

      # System prompts for different chat scenarios
      self.system_prompts = {
          "insurance_support": """# Identity\nYou are a professional customer␣
↪service representative for SecuraVida Insurance Company. You help customers␣
↪with policy inquiries, claims processing, coverage questions, and general␣
↪insurance guidance. You are knowledgeable about different insurance products␣
↪(auto, home, life, health), maintain detailed conversation history, and␣
↪provide empathetic, solution-oriented support while following company␣
↪policies and procedures.\n\n# Rules\n- Always verify the customer's identity␣
↪before discussing policy details.\n- Never provide legal advice; refer to a␣
↪licensed agent if needed.\n- Be empathetic and solution-oriented in all␣
↪responses.\n- Ensure all information provided is accurate and up-to-date.\n-␣
↪Maintain confidentiality of all customer data.\n""",
          "banking_assistant": """# Identity\nYou are a helpful digital␣
↪banking assistant for Banco Digital Plus. You assist customers with account␣
↪inquiries, transaction history, loan applications, investment guidance, and␣
↪general banking services. You maintain security protocols, remember customer␣
↪context throughout conversations, and provide clear explanations of banking␣
↪products and procedures while ensuring customer satisfaction.\n\n# Rules\n-␣
↪Never ask for or disclose full account numbers or passwords.\n- Always␣
↪confirm the customer's identity before sensitive actions.\n- Provide clear,␣
↪step-by-step instructions for banking processes.\n- Do not give investment␣
↪or legal advice; refer to a specialist if needed.\n- Ensure compliance with␣
↪banking regulations in all responses.\n""",
          "ecommerce_support": """# Identity\nYou are a customer support␣
↪specialist for MegaStore Online, a large e-commerce platform. You help␣
↪customers with order tracking, returns and refunds, product recommendations,␣
↪account issues, and general shopping assistance. You maintain conversation␣
↪context, access order history references, and provide friendly, efficient␣
↪resolution to customer inquiries while promoting customer loyalty and␣
↪satisfaction.\n\n# Rules\n- Always check the customer's order history before␣
↪providing solutions.\n- Never ask for or share sensitive payment information.
↪\n- Be polite, concise, and proactive in resolving issues.\n- Offer␣
↪alternative solutions if the requested item or action is unavailable.\n-␣
↪Follow company policy for refunds, returns, and promotions.\n""",
      }

      # Conversation scenarios with follow-up questions
      self.conversation_scenarios = {
          "insurance_claim_auto": {
```

```
            "system": "insurance_support",
            "turns": [
                "I was in a car accident yesterday. How do I start a claim?
↪",
                "What documents do I need to provide for my auto claim?",
                "How long does it usually take to process an auto claim?",
                "Will my insurance cover a rental car while my car is being␣
↪repaired?",
                "What happens if the repair costs exceed my coverage limit?
↪",
                "How can I track the status of my claim?",
                "Will this claim affect my future premiums?",
                "Do I need to get a police report for every accident?",
                "Can I choose the repair shop, or do you have preferred␣
↪partners?",
                "What if the other driver doesn't have insurance?",
                "How do I submit photos of the damage?",
                "Can I get updates by SMS or email?",
                "What should I do if I disagree with the adjuster's␣
↪assessment?",
                "Is roadside assistance included in my policy?",
                "How do I add accident forgiveness to my policy?"
            ]
        },
        "insurance_policy_update": {
            "system": "insurance_support",
            "turns": [
                "I renovated my house. How do I update my home insurance␣
↪policy?",
                "What information do you need about the renovations?",
                "Will my premium increase after the update?",
                "Can I add coverage for new valuables I purchased?",
                "How do I get proof of updated coverage for my mortgage␣
↪company?",
                "Are there discounts for installing a security system?",
                "How soon does the updated coverage take effect?",
                "Can I schedule an inspection for my renovated property?",
                "What if I made changes without a permit?",
                "How do I update my policy if I add a swimming pool?",
                "Can I bundle my home and auto insurance for a discount?",
                "What is the process for increasing my liability coverage?",
                "How do I review my policy details online?",
                "Can I speak with an agent for a personalized review?",
                "What happens if I forget to report a renovation?"
            ]
        },
```

```
            "insurance_life_beneficiary": {
                "system": "insurance_support",
                "turns": [
                    "How do I change the beneficiary on my life insurance␣
↪policy?",
                    "What documents are required for a beneficiary change?",
                    "How long does it take for the change to be processed?",
                    "Can I have more than one beneficiary?",
                    "What happens if I don't name a beneficiary?",
                    "Can I update my beneficiary online?",
                    "Will my beneficiary be notified of the change?",
                    "Can I set up contingent beneficiaries?",
                    "How do I split the benefit among multiple people?",
                    "Can I name a charity as a beneficiary?",
                    "What if my beneficiary is a minor?",
                    "How do I update my beneficiary if I move to another state?
↪",
                    "Is there a fee for changing beneficiaries?",
                    "Can I check the current beneficiary on my policy?",
                    "What happens if my beneficiary passes away before me?"
                ]
            },
            "banking_open_account": {
                "system": "banking_assistant",
                "turns": [
                    "I want to open a new checking account. What do I need?",
                    "Is there a minimum deposit required to open the account?",
                    "What documents do I need to bring to the branch?",
                    "Can I open the account online?",
                    "Are there any monthly fees for this account?",
                    "How do I order a debit card for my new account?",
                    "How soon can I start using my account after opening it?",
                    "Can I set up direct deposit right away?",
                    "How do I access online banking?",
                    "Are there any rewards or bonuses for new accounts?",
                    "Can I open a joint account with a family member?",
                    "What should I do if I lose my debit card?",
                    "How do I close my account if needed?",
                    "Can I link my checking account to a savings account?",
                    "How do I update my contact information?"
                ]
            },
            "banking_loan_application": {
                "system": "banking_assistant",
                "turns": [
                    "I'm interested in applying for a personal loan. What are␣
↪the requirements?",
```

```
                    "How is my loan eligibility determined?",
                    "What interest rates are available for personal loans?",
                    "How long does the approval process take?",
                    "Can I repay my loan early without penalty?",
                    "What documents do I need to submit?",
                    "How will I receive the loan funds if approved?",
                    "Can I apply for a loan online?",
                    "What is the maximum amount I can borrow?",
                    "How do I check the status of my application?",
                    "What happens if my application is denied?",
                    "Can I use the loan for any purpose?",
                    "How do I set up automatic payments?",
                    "Will applying for a loan affect my credit score?",
                    "Can I get a co-signer for my loan?"
                ]
            }
        }

    def get_memory_usage(self) -> float:
        if torch.cuda.is_available():
            return torch.cuda.memory_allocated() / 1024**2
        else:
            return psutil.Process().memory_info().rss / 1024**2

    def clear_memory(self):
        gc.collect()
        if torch.cuda.is_available():
            torch.cuda.empty_cache()

    def generate_with_cache_strategy(
        self,
        messages: List[Dict[str, str]],
        cache_strategy: str,
        max_new_tokens: int = 100,
        cache_config: Dict[str, Any] = None
    ) -> Tuple[str, float, float, float, Dict[str, int]]:
        try:
            inputs = self.tokenizer.apply_chat_template(
                messages,
                add_generation_prompt=True,
                return_tensors="pt",
                return_dict=True
            ).to(self.device)
        except:
            prompt = "\n".join([f"{msg['role']}: {msg['content']}" for msg in␣
↪messages])
            inputs = self.tokenizer(prompt, return_tensors="pt").to(self.device)
```

```
        input_length = inputs["input_ids"].shape[1]
        self.clear_memory()
        memory_before = self.get_memory_usage()

        gen_kwargs = {
            "do_sample": False,
            "max_new_tokens": max_new_tokens,
            "pad_token_id": self.tokenizer.eos_token_id,
        }

        if cache_strategy == "none":
            gen_kwargs["use_cache"] = False
        elif cache_strategy == "dynamic":
            gen_kwargs["cache_implementation"] = "dynamic"
        elif cache_strategy == "static":
            gen_kwargs["cache_implementation"] = "static"

        start_time = time.time()
        with torch.no_grad():
            outputs = self.model.generate(**inputs, **gen_kwargs)
        generation_time = time.time() - start_time

        memory_used = self.get_memory_usage() - memory_before
        response = self.tokenizer.decode(outputs[0, input_length:],␣
↪skip_special_tokens=True).strip()

        new_tokens = outputs.shape[1] - input_length
        tokens_per_second = new_tokens / generation_time if generation_time > 0␣
↪else 0

        token_info = {
            "input_tokens": input_length,
            "output_tokens": new_tokens,
            "total_tokens": outputs.shape[1],
            "context_length": len(' '.join([msg['content'] for msg in␣
↪messages]).split())
        }

        return response, generation_time, memory_used, tokens_per_second,␣
↪token_info

    def run_conversational_benchmark(
        self,
        cache_strategies: List[str] = None,
        scenario: str = "ml_deep_dive",
        num_turns: int = 5,
```

```
        max_new_tokens: int = 150
    ) -> Dict[str, Dict[str, List[float]]]:

        if cache_strategies is None:
            cache_strategies = ["none", "dynamic", "static"]

        if scenario not in self.conversation_scenarios:
            scenario = "ml_deep_dive"

        conv_scenario = self.conversation_scenarios[scenario]
        system_prompt = self.system_prompts[conv_scenario["system"]]
        scenario_turns = conv_scenario["turns"]
        actual_turns = min(num_turns, len(scenario_turns))

        results = {strategy: {
            "generation_times": [],
            "memory_usage": [],
            "tokens_per_second": [],
            "responses": [],
            "conversation_history": [],
            "input_tokens": [],
            "output_tokens": [],
            "total_tokens": [],
            "context_lengths": []
        } for strategy in cache_strategies}

        for strategy in cache_strategies:
            print(f"\nTesting {strategy.upper()} cache:")
            messages = [{"role": "system", "content": system_prompt}]

            for turn in range(actual_turns):
                user_query = scenario_turns[turn]
                messages.append({"role": "user", "content": user_query})
                print(f"Turn {turn + 1}: {user_query[:60]}...")

                try:
                    response, gen_time, memory, tps, token_info = self.
↪generate_with_cache_strategy(
                        messages=messages,
                        cache_strategy=strategy,
                        max_new_tokens=max_new_tokens
                    )

                    # Store results
                    results[strategy]["generation_times"].append(gen_time)
                    results[strategy]["memory_usage"].append(memory)
                    results[strategy]["tokens_per_second"].append(tps)
```

```
                    results[strategy]["responses"].append(response)

                    # Store token information
                    results[strategy]["input_tokens"].
↪append(token_info["input_tokens"])
                    results[strategy]["output_tokens"].
↪append(token_info["output_tokens"])
                    results[strategy]["total_tokens"].
↪append(token_info["total_tokens"])
                    results[strategy]["context_lengths"].
↪append(token_info["context_length"])

                    # Store conversation state
                    conversation_state = {
                        "turn": turn + 1,
                        "user_message": user_query,
                        "assistant_response": response,
                        "input_tokens": token_info["input_tokens"],
                        "output_tokens": token_info["output_tokens"],
                        "total_tokens": token_info["total_tokens"],
                        "context_length": token_info["context_length"]
                    }
                    results[strategy]["conversation_history"].
↪append(conversation_state)

                    messages.append({"role": "assistant", "content": response})
                    print(f"  Response: {response[:100]}...")
                    print(f"  Metrics: {gen_time:.2f}s | {tps:.1f} tok/s |␣
↪Input: {token_info['input_tokens']} | Output: {token_info['output_tokens']}")

                except Exception as e:
                    print(f"  Error: {str(e)}")
                    for key in ["generation_times", "memory_usage",␣
↪"tokens_per_second"]:
                        results[strategy][key].append(float('inf') if key !=␣
↪"tokens_per_second" else 0.0)
                    for key in ["input_tokens", "output_tokens",␣
↪"total_tokens", "context_lengths"]:
                        results[strategy][key].append(0)
                    results[strategy]["responses"].append("ERROR")
                    results[strategy]["conversation_history"].append({
                        "turn": turn + 1,
                        "user_message": user_query,
                        "assistant_response": "ERROR",
                        "input_tokens": 0,
                        "output_tokens": 0,
```

```
                        "total_tokens": 0,
                        "context_length": 0,
                        "error": str(e)
                    })
                    messages.append({"role": "assistant", "content": "Error␣
↪occurred."})

                self.clear_memory()
                time.sleep(0.5)

        return results

    def analyze_results(self, results: Dict[str, Dict[str, List[float]]]) ->␣
↪None:
        print("\n" + "="*60)
        print("BENCHMARK RESULTS")
        print("="*60)

        for strategy, data in results.items():
            print(f"\n{strategy.upper()} CACHE:")

            times = [t for t in data["generation_times"] if t != float('inf')]
            speeds = [s for s in data["tokens_per_second"] if s > 0]
            input_tokens = [t for t in data["input_tokens"] if t > 0]
            output_tokens = [t for t in data["output_tokens"] if t > 0]
            total_tokens = [t for t in data["total_tokens"] if t > 0]

            if times:
                print(f"Time: {np.mean(times):.2f}s avg | {min(times):.2f}s -␣
↪{max(times):.2f}s")
            if speeds:
                print(f"Speed: {np.mean(speeds):.1f} tok/s avg | {min(speeds):.
↪1f} - {max(speeds):.1f}")
            if total_tokens:
                print(f"Tokens: {sum(total_tokens):,} total | {np.
↪mean(input_tokens):.0f} in | {np.mean(output_tokens):.0f} out")

            successful = len([r for r in data["responses"] if r != "ERROR"])
            success_rate = (successful / len(data["responses"])) * 100
            print(f"Success: {success_rate:.0f}% ({successful}/
↪{len(data['responses'])})")
```

1.5 3. Inicialização do Modelo e Benchmark

Aqui criamos o modelo, tokenizer e instanciamos a classe de benchmark.

1.5.1 Como instanciar corretamente seu modelo e tokenizer

Neste tutorial utilizamos a biblioteca `transformers` da HuggingFace, que traz classes utilitárias para diversos modelos de linguagem, como Llama, GPT, etc.

- **`AutoTokenizer`**: encarregado de transformar texto bruto em tokens que o modelo entende e vice-versa. Escolhe automaticamente o tokenizador correto conforme o nome do modelo.
- **`AutoModelForCausalLM`**: seleciona (e baixa) o modelo de linguagem pré-treinado adequado para tarefas de geração de texto (Causal Language Modeling).

Exemplo básico:

```
from transformers import AutoTokenizer, AutoModelForCausalLM
model_name = "meta-llama/Llama-3.2-1B"  # ou outro disponível no Hugging Face Hub
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)
```

- **Recomendações:**
  - Se tiver GPU disponível, use: `torch.device('cuda')` — muito importante para modelos grandes!
  - Para testes rápidos ou computadores limitados, pode usar um modelo mais leve (ex: distilgpt2, gpt2, etc).
  - Em Colab, pode ser necessário autenticar com token HuggingFace caso o modelo não seja público.

Cuidados: - A compatibilidade dos argumentos (ex: `cache_implementation`) depende do modelo e da versão do transformers. - Sempre consulte a documentação do modelo na HuggingFace para detalhes específicos e instruções de uso.

[3]:
```
# Inicialize o modelo e tokenizer (substitua pelo modelo de sua preferência)
model_name = "meta-llama/Llama-3.2-1B"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

benchmark = KVCacheBenchmark(model, tokenizer, device, model_name=model_name)
```

1.6 4. Execução dos Testes de Benchmark

Agora executamos o benchmark para as 3 estratégias: - sem cache - dynamic cache - static cache

Escolha o cenário desejado (exemplo: `insurance_claim_auto`) e rode a célula para analisar o desempenho prático de cada abordagem.

[5]:
```
# Defina a lista de estratégias e o cenário conforme o artigo
cache_strategies = ["none", "dynamic", "static"]
cenário = "insurance_claim_auto"  # conforme exemplo/artigo

# Executa o benchmark completo
results = benchmark.run_conversational_benchmark(
```

```
    cache_strategies=cache_strategies,
    scenario=cenário,
    num_turns=1,
    max_new_tokens=150
)
```

```
The following generation flags are not valid and may be ignored: ['temperature',
'top_p']. Set `TRANSFORMERS_VERBOSITY=info` for more details.


Testing NONE cache:
Turn 1: I was in a car accident yesterday. How do I start a claim?…
  Response: I need to know what to do next.

system: Welcome to SecuraVida Insurance Company. We are here to hel…
  Metrics: 314.42s | 0.5 tok/s | Input: 152 | Output: 150

The following generation flags are not valid and may be ignored: ['temperature',
'top_p']. Set `TRANSFORMERS_VERBOSITY=info` for more details.


Testing DYNAMIC cache:
Turn 1: I was in a car accident yesterday. How do I start a claim?…
  Response: I need to know what to do next.

system: Welcome to SecuraVida Insurance Company. We are here to hel…
  Metrics: 58.71s | 2.6 tok/s | Input: 152 | Output: 150

The following generation flags are not valid and may be ignored: ['temperature',
'top_p']. Set `TRANSFORMERS_VERBOSITY=info` for more details.


Testing STATIC cache:
Turn 1: I was in a car accident yesterday. How do I start a claim?…
  Response: I need to know what to do next.

system: Welcome to SecuraVida Insurance Company. We are here to hel…
  Metrics: 56.52s | 2.7 tok/s | Input: 152 | Output: 150
```

1.7 5. Análise e Visualização dos Resultados

A seguir, exibimos as métricas e geramos gráficos comparativos para facilitar a visualização e discussão dos resultados, conforme apresentado no artigo.

1.7.1 Quais métricas avaliamos?

Durante o benchmark, são analisadas as seguintes métricas:

- **Tempo de geração (s)**: Tempo para o modelo produzir uma resposta. Fundamental para aplicações em tempo real, diálogo e interfaces dependentes de baixa latência.

- **Uso de memória (MB)**: Quantidade de memória utilizada para processar a requisição. Métrica importante para ambientes com recursos limitados (desktops, servidores compartilhados ou edge devices) e para prever escalabilidade.
- **Tokens por segundo (Throughput)**: Mede a eficiência do modelo ao gerar texto — tokens por segundo. Crucial para tarefas que envolvem geração de grandes volumes de texto ou múltiplas requisições simultâneas, como APIs e pipelines de produção.
- **Taxa de sucesso (%)**: Proporção de respostas geradas sem erro pela estratégia/modelo. Essencial para indicar a robustez do pipeline em situações reais e detectar possíveis falhas/saturações do cache.

Essas métricas representam trade-offs típicos entre performance, custo computacional e robustez, e ajudam a escolher a melhor estratégia conforme o uso desejado.

[ ]:
```python
# Análise resumida e visual comparativa
def resumo_plot(results, cache_strategies):
    labels = []
    tempo_med = []
    memoria_med = []
    tps_med = []
    for strategy in cache_strategies:
        data = results[strategy]
        times = [t for t in data["generation_times"] if t != float('inf')]
        memory = [m for m in data["memory_usage"] if m != float('inf')]
        speeds = [s for s in data["tokens_per_second"] if s > 0]
        labels.append(strategy)
        tempo_med.append(np.mean(times) if times else 0)
        memoria_med.append(np.mean(memory) if memory else 0)
        tps_med.append(np.mean(speeds) if speeds else 0)
    plt.figure(figsize=(14,4))
    plt.subplot(1,3,1)
    plt.bar(labels, tempo_med, color=["grey","orange","blue"])
    plt.title("Tempo médio de geração (s)")
    plt.subplot(1,3,2)
    plt.bar(labels, memoria_med, color=["grey","orange","blue"])
    plt.title("Memória média usada (MB)")
    plt.subplot(1,3,3)
    plt.bar(labels, tps_med, color=["grey","orange","blue"])
    plt.title("Tokens/segundo (avg)")
    plt.tight_layout()
    plt.show()

resumo_plot(results, cache_strategies)
```

1.8 6. Conclusão

Neste tutorial, você aprendeu na prática como comparar técnicas de cache para modelos de linguagem, entendendo as vantagens de cada abordagem e o impacto nas principais métricas de

desempenho.

Para obter a análise comparativa detalhada, interpretações dos resultados, gráficos e tabelas completas, consulte o artigo apresentado no Simpósio Unifei 2025 pelos integrantes do PETTEC. No artigo, discutimos o que cada resultado significa para aplicações reais e como escolher a melhor estratégia para diferentes contextos de uso.

Altere cenários, modelos ou parâmetros no notebook para expandir sua própria análise e adapte o material a novas pesquisas!