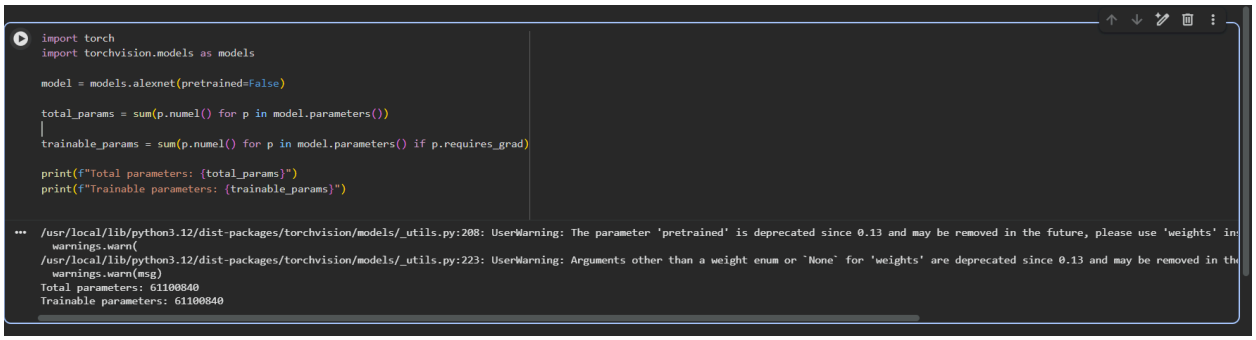
Number of Parameters in AlexNet:



```python
import torch
import torchvision.models as models

model = models.alexnet(pretrained=False)

total_params = sum(p.numel() for p in model.parameters())

trainable_params = sum(p.numel() for p in model.parameters() if p.requires_grad)

print(f"Total parameters: {total_params}")
print(f"Trainable parameters: {trainable_params}")
```

```
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the future, please use 'weights' ins
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed in th
  warnings.warn(msg)
Total parameters: 61100840
Trainable parameters: 61100840
```

CUDA

Compute Unified Device Architecture (CUDA) is a parallel computing platform and programming model developed by NVIDIA that enables developers to leverage the massive computational power of Graphics Processing Units (GPUs) for general-purpose computing. By exposing GPU hardware capabilities through a high-level programming interface, CUDA has transformed performance-intensive domains such as scientific computing, machine learning, image processing, and real-time simulation. This paper presents an overview of CUDA's architecture, programming model, memory hierarchy, and its impact on modern high-performance computing.

Traditional Central Processing Units (CPUs) are optimized for sequential execution and low-latency task switching, making them less efficient for data-parallel workloads. GPUs, originally designed for graphics rendering, consist of thousands of smaller, efficient cores capable of executing many operations concurrently. CUDA bridges the gap between GPU hardware and general-purpose programming, allowing developers to write parallel code using familiar languages such as C, C++, and Python. Since its introduction in 2007, CUDA has become a cornerstone of GPU-accelerated computing.

The CUDA architecture is based on a heterogeneous computing model, where the CPU (host) and GPU (device) work together. The host manages control flow and memory allocation, while the device executes parallel kernels. GPUs are organized into Streaming Multiprocessors (SMs), each containing multiple CUDA cores capable of executing threads concurrently.

CUDA follows a Single Instruction, Multiple Threads (SIMT) execution model. Threads are grouped into warps (typically 32 threads), which execute the same instruction simultaneously on different data elements. Efficient performance depends on minimizing warp divergence and maximizing occupancy of SMs.

In CUDA, parallel code is written as kernels, which are functions executed on the GPU. A kernel launch specifies a hierarchy of execution:

- Threads: the smallest unit of execution
- Blocks: groups of threads that can cooperate via shared memory
- Grids: collections of blocks executing a kernel

This hierarchical model allows scalability across different GPU architectures. CUDA provides built-in variables such as threadIdx, blockIdx, and blockDim to uniquely identify threads and map them to data elements.

Efficient memory usage is critical for CUDA performance. CUDA exposes multiple memory types:

- Global memory: large but high-latency
- Shared memory: low-latency, shared among threads in a block
- Registers: fastest, private to each thread

- Constant and texture memory: cached, read-only memory spaces

Optimizing memory access patterns, such as coalesced global memory access and effective use of shared memory, is essential for achieving high throughput.

CUDA has enabled significant advancements across multiple domains. In scientific computing, CUDA accelerates simulations in physics, chemistry, and climate modeling. In machine learning, CUDA forms the backbone of frameworks such as TensorFlow and PyTorch, enabling efficient training of deep neural networks. Other applications include video encoding, financial modeling, bioinformatics, and autonomous systems.

CUDA has revolutionized parallel computing by making GPU acceleration accessible to a broad range of developers. Its well-defined programming model, scalable architecture, and rich ecosystem of libraries have made it a dominant platform for high-performance and data-parallel applications. As GPU architectures continue to evolve, CUDA remains a critical tool for addressing the growing demand for computational power in modern computing workloads.

Agentic Browsers: Definition, Architecture, and Use Cases

Agentic browsers represent a new paradigm in web interaction, combining large language models (LLMs), autonomous decision-making, and traditional web browsing to complete complex tasks on behalf of users. Unlike conventional browsers that rely solely on human input, agentic browsers use AI agents to autonomously interpret goals, navigate websites, and execute multi-step workflows. This paper defines agentic browsers, examines their core mechanisms, surveys practical use cases, and discusses both the potential and challenges of this emerging technology.

Web browsers have long served as passive interfaces for users to access content on the internet. Traditional browsing requires users to manually search, evaluate results, click links, fill out forms, and complete transactions. Agentic browsers, by contrast, shift this model by embedding autonomous AI agents within the browser environment, allowing users to issue high-level commands such as *“find the cheapest flight”* or *“book dinner for Friday night”* and have the system execute all required steps without continuous human intervention

An agentic browser is a web client controlled by an AI agent that can not only read and render web pages but *interact* with web elements, reason over content, and perform actions like link navigation, button clicking, form submission, and purchase processing based on user goals . Core components typically include:

- Intent Interpretation: Natural language processing to understand user objectives.
- Website Analysis: Parsing of HTML/CSS and interactive elements to determine possible actions.
- Action Planning and Execution: Breaking down goals into steps and executing them autonomously with browser automation techniques.
- Memory and Adaptation: Retention of context across tasks to improve performance over time .

This combination of capabilities transforms the browser from a passive *viewer* into an *executor* of tasks, leveraging LLMs for reasoning and dynamic navigation logic that handles unexpected web layouts and conditional workflows.

Agentic browsers excel in automating repetitive or multi-step tasks that traditionally require significant manual effort. Examples include:

- Booking Travel and Reservations: Searching airline and hotel sites, comparing prices, and completing bookings based on user preferences.

- Form Filling and Transactions: Auto-filling complex forms, processing orders, and completing purchases with minimal user input.

- Research and Summarization: Navigating multiple sources, extracting key data, synthesizing findings, and producing summaries or structured reports.
  These automation tasks reduce cognitive load and free users' time for higher-order analysis, enhancing productivity across personal and business domains .

Agentic browsers can autonomously extract structured data from multiple websites, enabling:

- Market Research: Collecting competitor pricing and product information.

- Trend Analysis: Aggregating and correlating data from diverse sources.

- Content Aggregation: Compiling information for reporting, dashboards, or academic research.

Such capabilities suit analysts, marketers, and researchers who require large-scale web data without manual scraping or custom scripting .

As agents learn from interactions, they can adapt to user preferences, delivering personalized task execution patterns. For instance:

- Customized Shopping Assistants: Automatically locating preferred brands or discounted items.

- Continuous Monitoring: Tracking price changes, availability of tickets or appointments, and notifying or acting when conditions are met.

These use cases demonstrate how agentic browsers move beyond static search to *action-oriented web engagement*.

While agentic browsers promise efficiency gains, they introduce significant challenges:

- Security Vulnerabilities: Autonomous execution increases exposure to prompt injection attacks and malicious instructions embedded in web content, potentially leading to data leakage or unintended actions .

- Loss of Control: Users may cede oversight as agents act autonomously, sometimes with insufficient contextual understanding.

- Privacy Concerns: Agents require access to sensitive data, raising concerns about credential handling and data retention.
  These risks have prompted industry advisories and the development of security adjustments such as human-in-the-loop authentication mechanisms for sensitive operations.

Agentic browsers constitute an emerging class of AI-embedded web technologies capable of transforming web interaction from reactive content browsing to proactive task execution. Their ability to perform multi-step workflows, automate research and data extraction, and personalize interactions holds considerable promise for productivity, business intelligence, and consumer convenience. However, realizing these benefits at scale will require rigorous security frameworks, transparent user control mechanisms, and robust privacy protections. As adoption grows, the evolution of agentic browsing will likely reshape how humans and machines collaborate on web-based tasks.