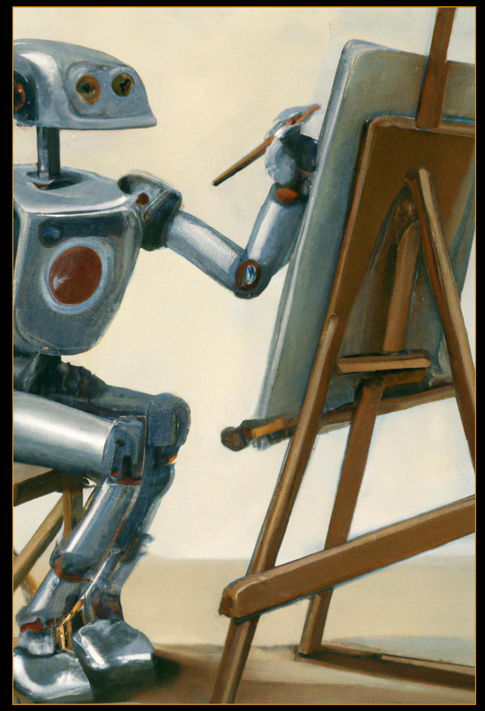
NVIDIA
Accelerating Research

Paul Graham - Senior Solutions Architect

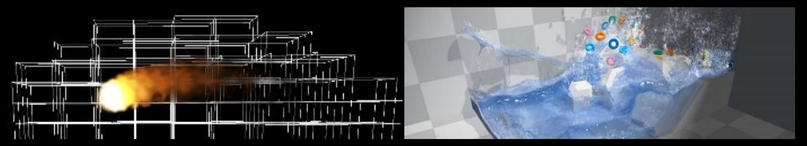
University of Oxford CUDA Summer School

23rd July 2026

NVIDIA



GPU Computing



Computer Graphics

Artificial Intelligence

GH100 GPU Architecture

https://resources.nvidia.com/en-us-tensor-core/gtc22-whitepaper-hopper

GH100 GPU A

https://resources.nvi r-hopper

GH100 GPU A

https://resources.nvi r-hopper

- 132 of these in the H100 SXM5 GPU
  - (224 in Rubin!)
- over 16000 FP32 cores in total
- we usually want number of threads >> num cores
- so we need ***a lot*** of threads!

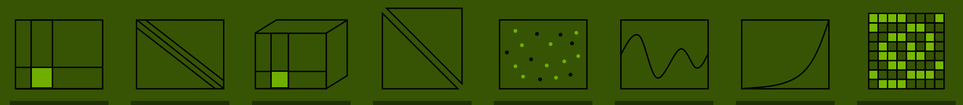
The CUDA Platform

Target the abstraction layer that works best for your application

Developer & Application Ecosystem

- Frameworks: PyTorch, TensorFlow, Jax, Modulus, Triton, ...
- SDKs: Medical Devices, Energy, Autonomous Vehicles, ...

NVIDIA Accelerated Libraries



cuBLAS | cuSPARSE | cuTENSOR | cuSOLVER | cuRAND | cuFFT | Math API | NPP

Parallel Languages

OpenACC	OpenMP	Standard C++	Python	Julia	MATLAB	...
CUDA FORTRAN	CUDA C++	OpenCL	Ada	Haskell	R	...

Compilation Stack

- NVVM / LLVM IR
- PTX Assembly ISA

20 Years of CUDA

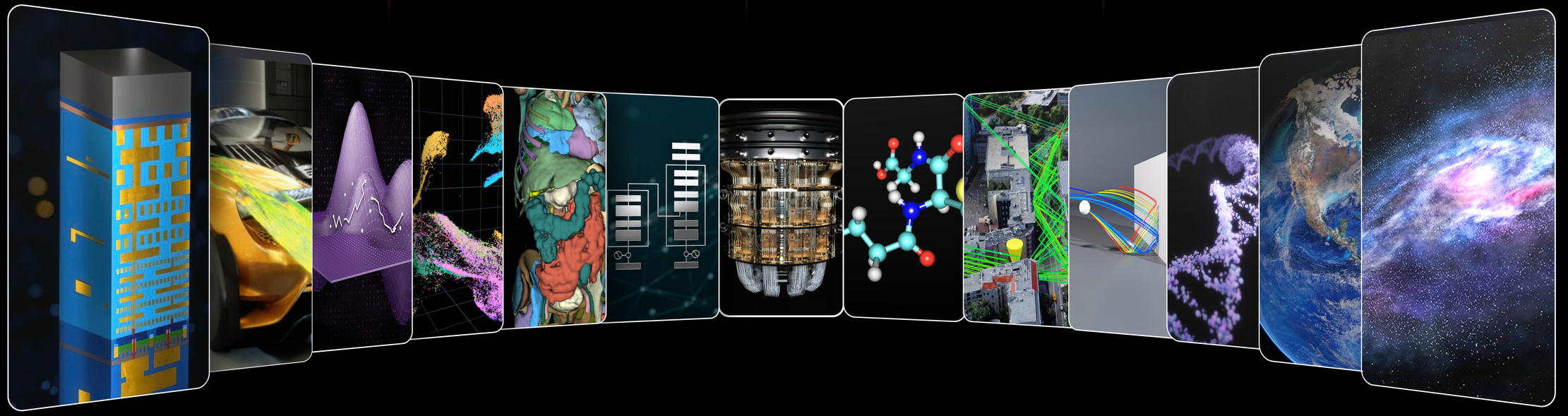
Every Industry | One Architecture | CUDA Everywhere

7M
Developers

600M
Downloads

6K
Accelerated Applications

1K
SDKs and Models



cuLitho
Computational Lithography

cuDSS
cuSPARSE
cuFFT
AmgX
Computer-Aided Engineering

cuOpt
Decision Optimization

cuDF
cuVS
cuML
cuGraph
Data Science and Processing

MONAI
Medical Imaging

Megatron
Dynamo
NIXL
cuDNN
CUTLASS
Deep Learning

cuQuantum
CUDA-Q
Quantum Computing

cuEST
cuEquivariance
cuTensor
ALCHEMI
Quantum Chemistry

Aerial
Sionna
Holoscan
DAQUIRI
5G/6G Signal Processing

Warp
Physics

Parabricks
Genomics

Earth-2
Weather Analytics

cuPyNumeric
cuPhoton
Numerical Computing

NVIDIA

New things in CUDA

Hierarchy of Execution Models

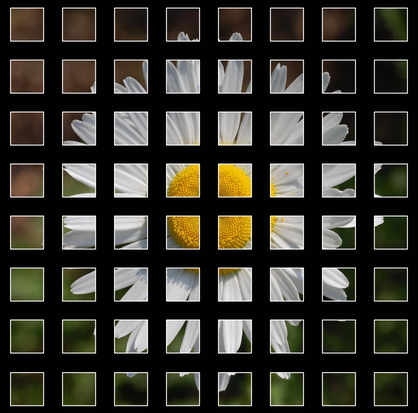
Grid-level models

Easy acceleration of
bulk data operations

Compiler maps
data onto both
blocks and threads

Block-level models

Productivity & performance
for array-based programs

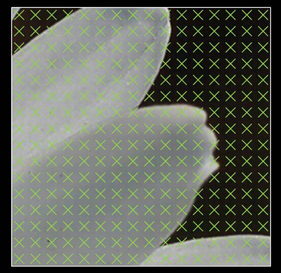


Application maps
data onto blocks

Compiler maps
blocks onto threads

Thread-level models

Full control for
all parallel workloads

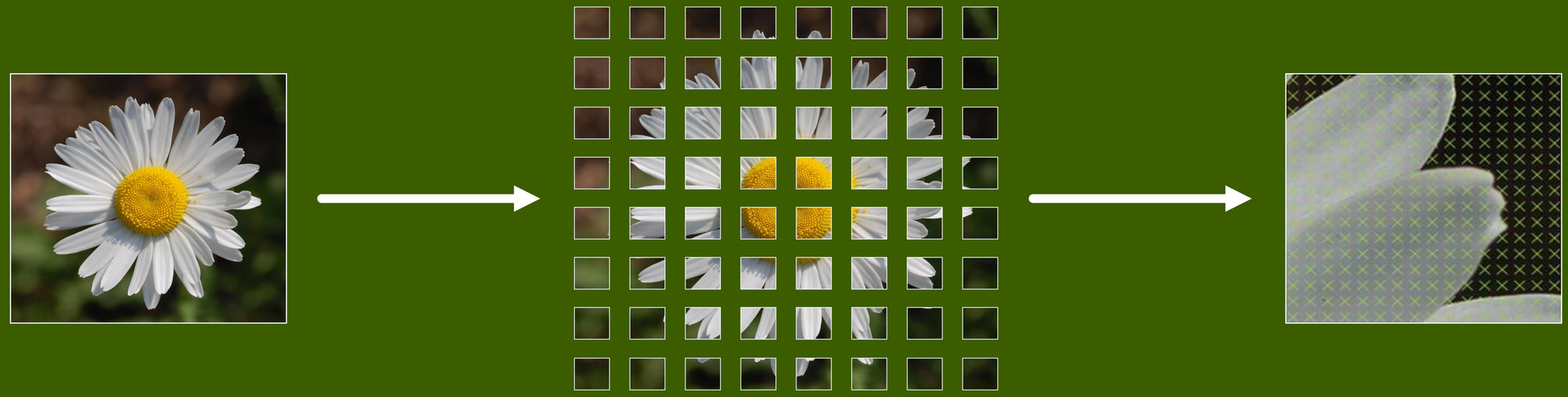


Application maps
data onto both
blocks and threads

CUDA SIMT User Responsibility

Split work into blocks
Divide data into tiles

Map blocks & tiles onto threads



Memory movement strategy?

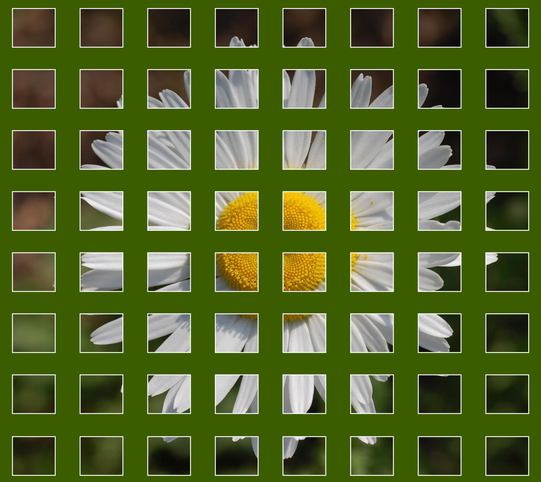
Compute strategy?

Tile memory layout?

Tile storage?

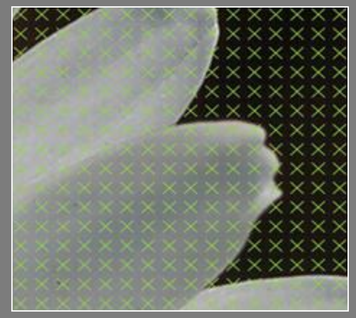
CUDA Tile User Responsibility

Split work into blocks
Divide data into tiles



System Responsibility

Map blocks & tiles onto threads



Memory movement strategy?

Compute strategy?

Tile memory layout?

Tile storage?

cuTile: A Python front-end for CUDA Tile

cuTile by example

```
import cuda.tile as ct

@ct.func
def softmax(I, O, r):
  c = ct.load(I, (0, ct.bid(0)), (I.shape[0], r))


  max = ct.max(c, axis=0, keepdims=True)
  num = ct.exp(c - max)
  den = ct.sum(num, axis=0, keepdims=True)
  smax = num / den


  ct.store(O, (0, ct.bid(0)), smax)
```

Exposes CUDA Tile as a NumPy-like array programming model.

https://developer.nvidia.com/cuda/tile - also C++ support

Naïve CUDA Python SIMT

```
@cuda.device.kernel
def matmul(A, B, C):
  TPB = cuda.blockDim.x
  BPG = cuda.gridDim.x
  x, y = cuda.grid(2)
  tx = cuda.threadIdx.x
  ty = cuda.threadIdx.y

  sA = cuda.shared.array((TPB, TPB), float32)
  sB = cuda.shared.array((TPB, TPB), float32)

  tmp = float32(0.)
  for i in range(BPG):
    sA[ty, tx] = 0
    sB[ty, tx] = 0
    if y < A.shape[0] and (tx+I*TPB) < A.shape[1]:
      sA[ty, tx] = A[y, tx + i * TPB]
    if x < B.shape[1] and (ty+i*TPB) < B.shape[0]:
      sB[ty, tx] = B[ty + i * TPB, x]

    cuda.syncthreads()
    for j in range(TPB):
      tmp += sA[ty, j] * sB[j, tx]
    cuda.syncthreads()

  if y < C.shape[0] and x < C.shape[1]:
    C[y, x] = tmp
```

cuTile Python

```
import cuda.tile as ct

@ct.kernel
def matmul(A: ct.Array,
           B: ct.Array,
           C: ct.Array,
           ts: ct.Constant[ct.Shape]):
  sum = ct.zeros(ts[0:1], A.dtype)
  (x, y) = (ct.bid(0), ct.bid(1))

  pA = ct.view.partition(A, (ts[0], ts[2]))
  pB = ct.view.partition(B, (ts[2], ts[1]))
  for k in range(pA.shape[1]):
    sum = ct.mma(pA[x, k], pB[k, y], sum)

  ct.store(C, (x, y), sum)
```

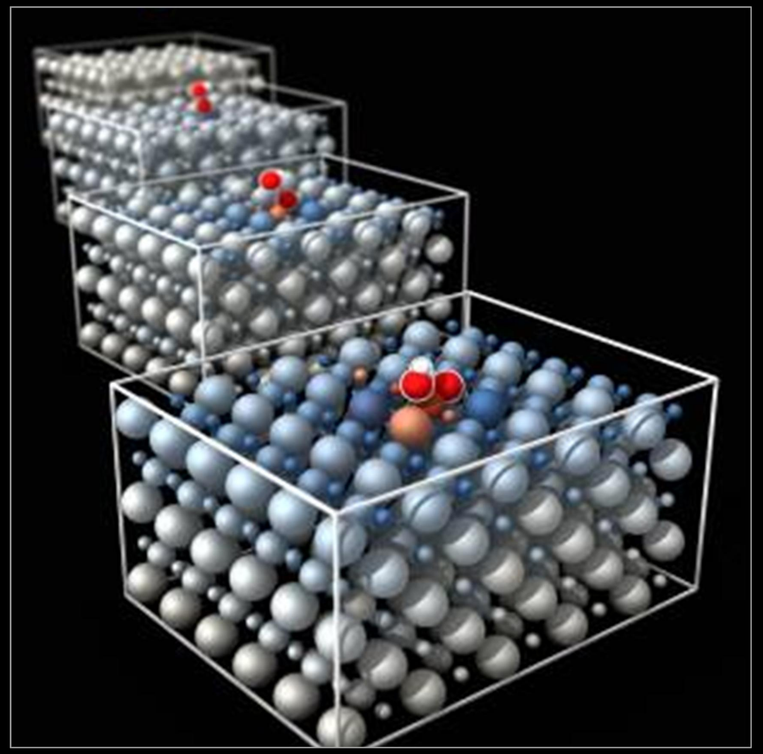
Speed-Of-Light Non-Portable CUDA C++ SIMT

```
using namespace cute;

template <class SharedStorage,
          class ATensor, class BTensor, class CTensor, class DTensor,
          class MmaTiler_MNK, class TiledMMA, class ClusterShape_MNK,
          class TmaAtomA, class TmaAtomB,
          class Alpha, class Beta>
__global__ static
void
matmul(ATensor mA,
           BTensor mB,
           CTensor mC,
           DTensor mD,
           MmaTiler_MNK mma_tiler,
           TiledMMA tiled_mma,
           ClusterShape_MNK cluster_shape,
           CUTE_GRID_CONSTANT TmaAtomA const tma_atom_A,
           CUTE_GRID_CONSTANT TmaAtomB const tma_atom_B,
           Alpha alpha, Beta beta)
{
  Layout cluster_layout_vmnk = tiled_divide(make_layout(cluster_shape),
    make_tile(typename TiledMMA::AtomThrID{}));

  auto mma_coord_vmnk = make_coord(blockIdx.x % size<0>(cluster_layout_vmnk),
                                   blockIdx.x / size<0>(cluster_layout_vmnk),
                                   blockIdx.y,
                                   _);

  auto mma_coord = select<1,2,3>(mma_coord_vmnk);
  Tensor gA = local_tile(mA, mma_tiler, mma_coord, Step<_1, X,_1>{});
  Tensor gB = local_tile(mB, mma_tiler, mma_coord, Step< X,_1,_1>{});
  Tensor gC = local_tile(mC, mma_tiler, mma_coord, Step<_1,_1, X>{});
  Tensor gD = local_tile(mD, mma_tiler, mma_coord, Step<_1,_1, X>{});
  __syncthreads();

  extern __shared__ char shared_memory[];
  SharedStorage& shared_storage =
    *reinterpret_cast<SharedStorage*>(shared_memory);

  Tensor tCsA = shared_storage.tensor_sA();
  Tensor tCsB = shared_storage.tensor_sB();

  auto mma_v = get<0>(mma_coord_vmnk);
  ThrMMA cta_mma = tiled_mma.get_slice(mma_v);
  Tensor tCgA = cta_mma.partition_A(gA);
  Tensor tCgB = cta_mma.partition_B(gB);
  Tensor tCgC = cta_mma.partition_C(gC);
  Tensor tCgD = cta_mma.partition_C(gD);
  __syncthreads();

  Tensor tCrA = cta_mma.make_fragment_A(tCsA);
  Tensor tCrB = cta_mma.make_fragment_B(tCsB);

  Tensor tCtAcc = cta_mma.make_fragment_C(tCgC);

  uint32_t elect_one_thr  = cute::elect_one_sync();
  uint32_t elect_one_warp = (threadIdx.x / 32 == 0);

  using TmemAllocator = cute::TMEM::Allocator1Sm;
  TmemAllocator tmem_allocator{};

  if (elect_one_warp) {
    tmem_allocator.allocate(TmemAllocator::Sm100TmemCapacityColumns,
      &shared_storage.tmem_base_ptr);
  }

  __syncthreads();
  tCtAcc.data() = shared_storage.tmem_base_ptr;
  __syncthreads();

  auto [tAgA, tAsA] = tma_partition(tma_atom_A, Int<0>{}, Layout<_1>{},
    group_modes<0,3>(tCsA), group_modes<0,3>(tCgA));

  auto [tBgB, tBsB] = tma_partition(tma_atom_B, Int<0>{}, Layout<_1>{},
    group_modes<0,3>(tCsB), group_modes<0,3>(tCgB));

  __syncthreads();

  if (elect_one_warp && elect_one_thr) {
    cute::initialize_barrier(shared_storage.mma_barrier, /* num_ctas */ 1);
    cute::initialize_barrier(shared_storage.tma_barrier, /* num_threads */ 1);
  }

  int mma_barrier_phase_bit = 0;
  int tma_barrier_phase_bit = 0;
  __syncthreads();

  tiled_mma.accumulate_ = UMMA::ScaleOut::Zero;

  for (int k_tile = 0; k_tile < size<3>(tCgA); ++k_tile)
  {
    if (elect_one_warp && elect_one_thr) {
      int tma_transaction_bytes = sizeof(make_tensor_like(tAsA))
                                + sizeof(make_tensor_like(tBsB));
      cute::set_barrier_transaction_bytes(shared_storage.tma_barrier,
                                          tma_transaction_bytes);
      copy(tma_atom_A.with(shared_storage.tma_barrier), tAgA(_,k_tile), tAsA);
      copy(tma_atom_B.with(shared_storage.tma_barrier), tBgB(_,k_tile), tBsB);
    }

    cute::wait_barrier(shared_storage.tma_barrier, tma_barrier_phase_bit);
    tma_barrier_phase_bit ^= 1;

    if (elect_one_warp) {
      for (int k_block = 0; k_block < size<2>(tCrA); ++k_block) {
        gemm(tiled_mma, tCrA(_,_,k_block), tCrB(_,_,k_block), tCtAcc);
        tiled_mma.accumulate_ = UMMA::ScaleOut::One;
      }
      cutlass::arch::umma_arrive(&shared_storage.mma_barrier);
    }
    cute::wait_barrier(shared_storage.mma_barrier, mma_barrier_phase_bit);
    mma_barrier_phase_bit ^= 1;
  }

  TiledCopy tiled_t2r_copy = make_tmem_copy(SM100_TMEM_LOAD_32dp32b1x{}, tCtAcc);
  ThrCopy   thr_t2r_copy   = tiled_t2r_copy.get_slice(threadIdx.x);

  Tensor tDgC = thr_t2r_copy.partition_D(tCgC);
  Tensor tDrC = make_fragment_like(tDgC);
  copy(tDgC, tDrC);

  Tensor tDtAcc = thr_t2r_copy.partition_S(tCtAcc);
  Tensor tDgD   = thr_t2r_copy.partition_D(tCgD);

  using AccType = typename decltype(tCtAcc)::value_type;
  Tensor tDrAcc = make_tensor<AccType>(shape(tDgD));

  copy(tiled_t2r_copy, tDtAcc, tDrAcc);

  axpby(alpha, tDrAcc, beta, tDrC);
  copy(tDrC, tDgD);

  __syncthreads();

  if (elect_one_warp) {
    tmem_allocator.release_allocation_lock();
    tmem_allocator.free(shared_storage.tmem_base_ptr,
      TmemAllocator::Sm100TmemCapacityColumns);
  }
}
```

cuTile Python

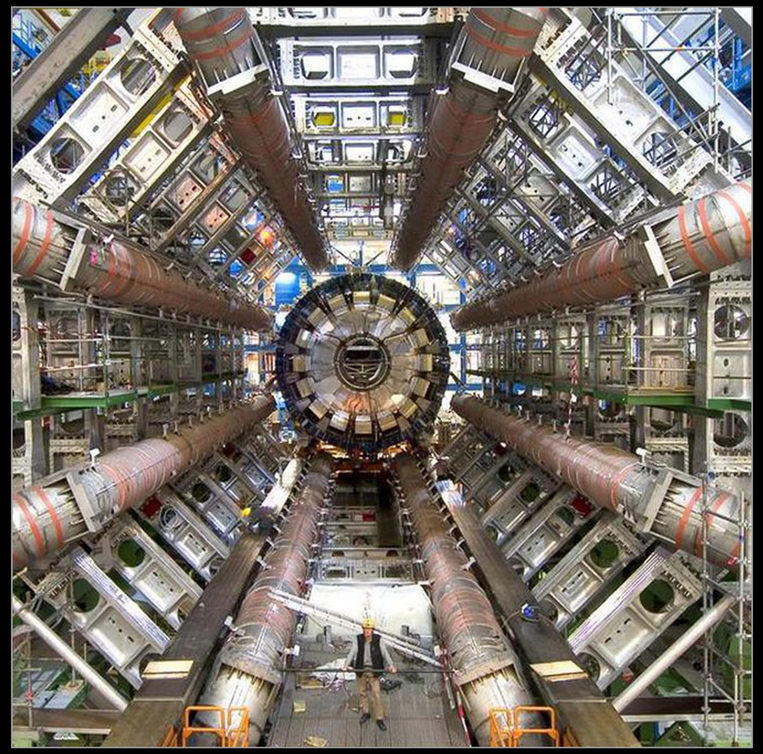
```
import cuda.tile as ct

@ct.kernel
def matmul(A: ct.Array,
           B: ct.Array,
           C: ct.Array,
           ts: ct.Constant[ct.Shape]):
  sum = ct.zeros(ts[0:1], A.dtype)
  (x, y) = (ct.bid(0), ct.bid(1))

  pA = ct.view.partition(A, (ts[0], ts[2]))
  pB = ct.view.partition(B, (ts[2], ts[1]))
  for k in range(pA.shape[1]):
    sum = ct.mma(pA[x, k], pB[k, y], sum)

  ct.store(C, (x, y), sum)
```

New Libraries and Frameworks



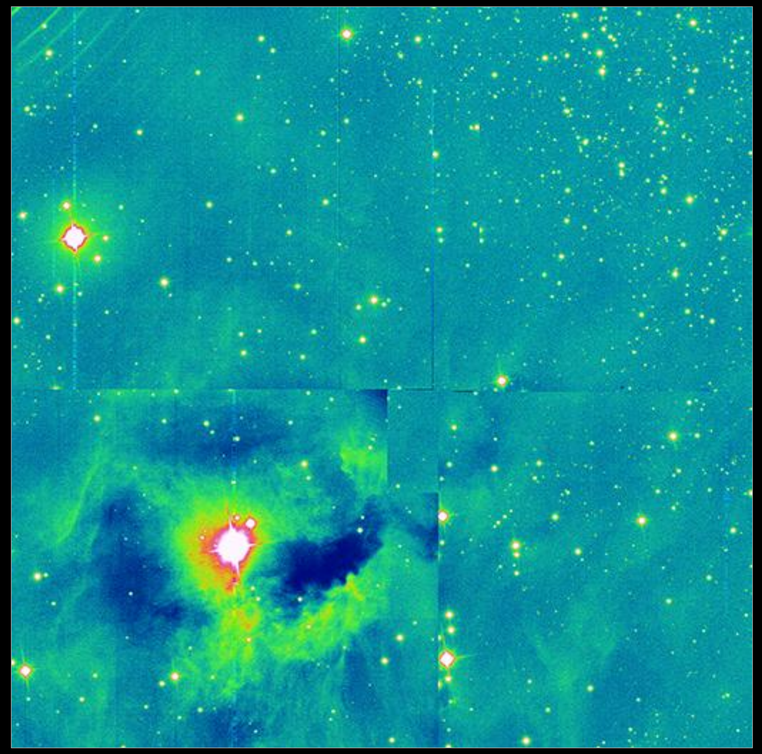
ALCHEMI

Up to 50x acceleration in magnetic materials screening using BGR and VASP microservice



DAQIRI

Connecting sensors to real-time AI inferencing and data processing at 100s of Gbps



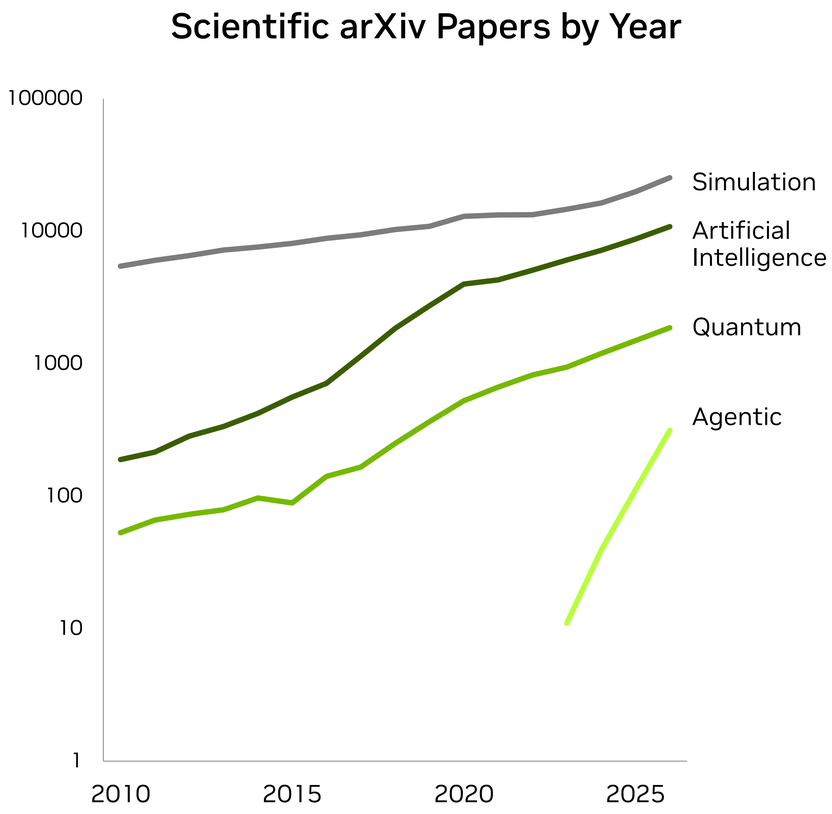
cuPhoton

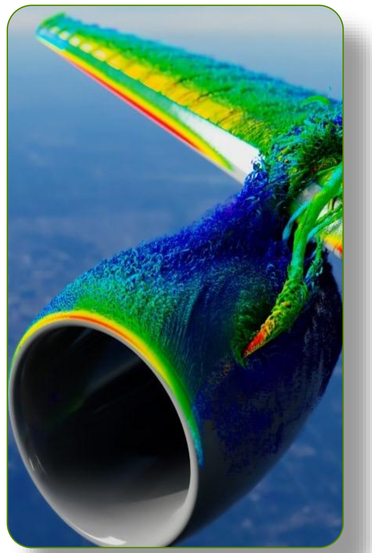
Accelerates image loading/reading by up to 15,000x, and processing and analysis by up to 8,000x

The future of simulation

Scientific Computing at an Inflection Point

From One to Four Pillars of Scientific Computing

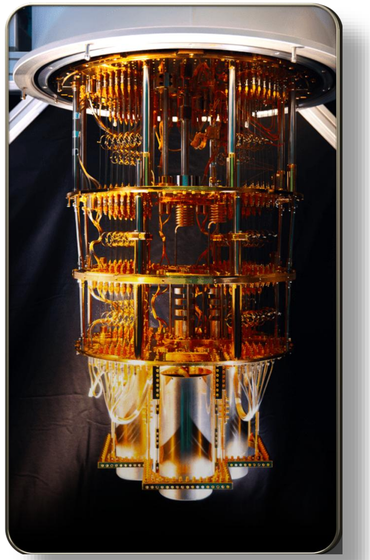




Simulation

AI

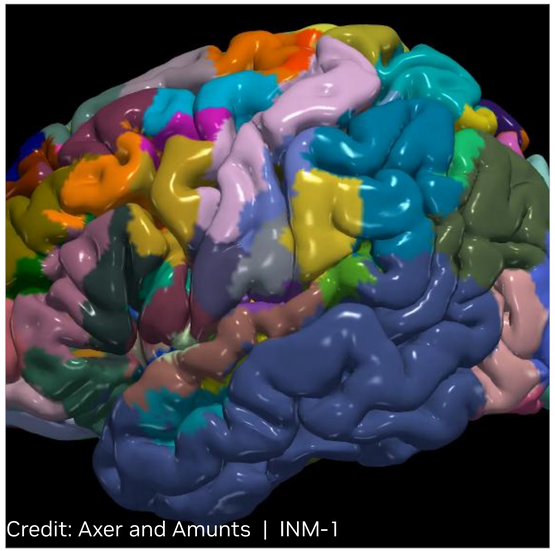
Agentic



Quantum

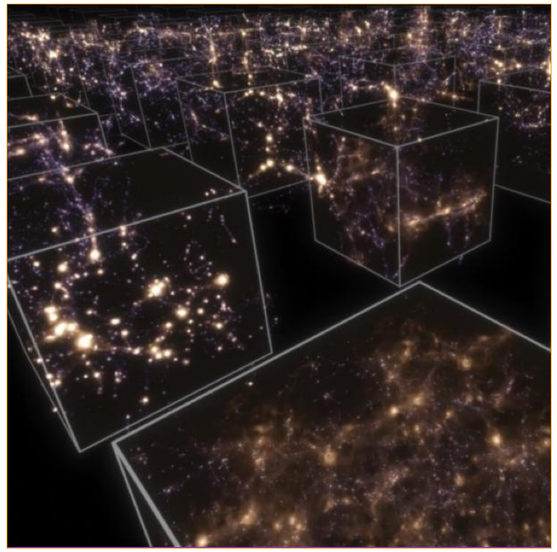
Jülich JUPITER Supercharging Sciences

Europe’s largest supercomputer powering discovery in healthcare, quantum, telecommunications and climate



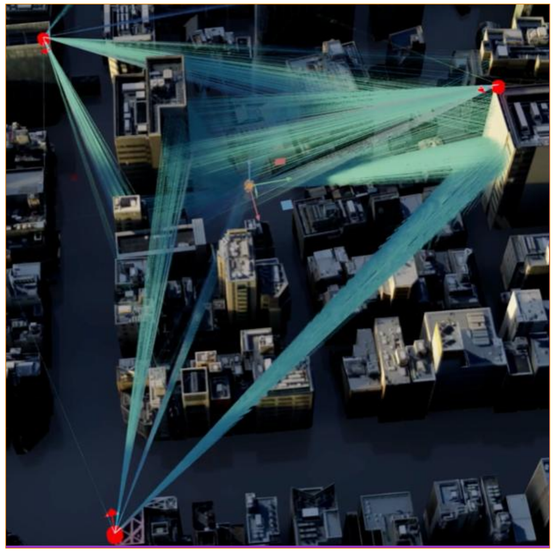
First 3-D Atlas for the Human Brain

An intelligent agent for brain research



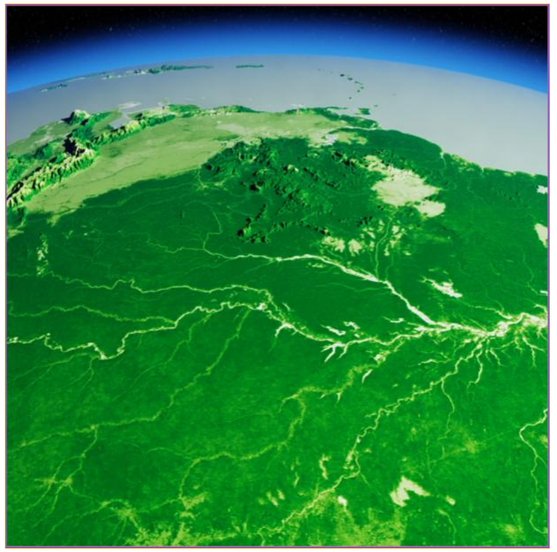
Quantum Simulation

State-vector simulation of
Shor’s algorithm at 50 qubits



6G Telecommunications

Neuromorphic computing at Ericcson



Climate Simulation

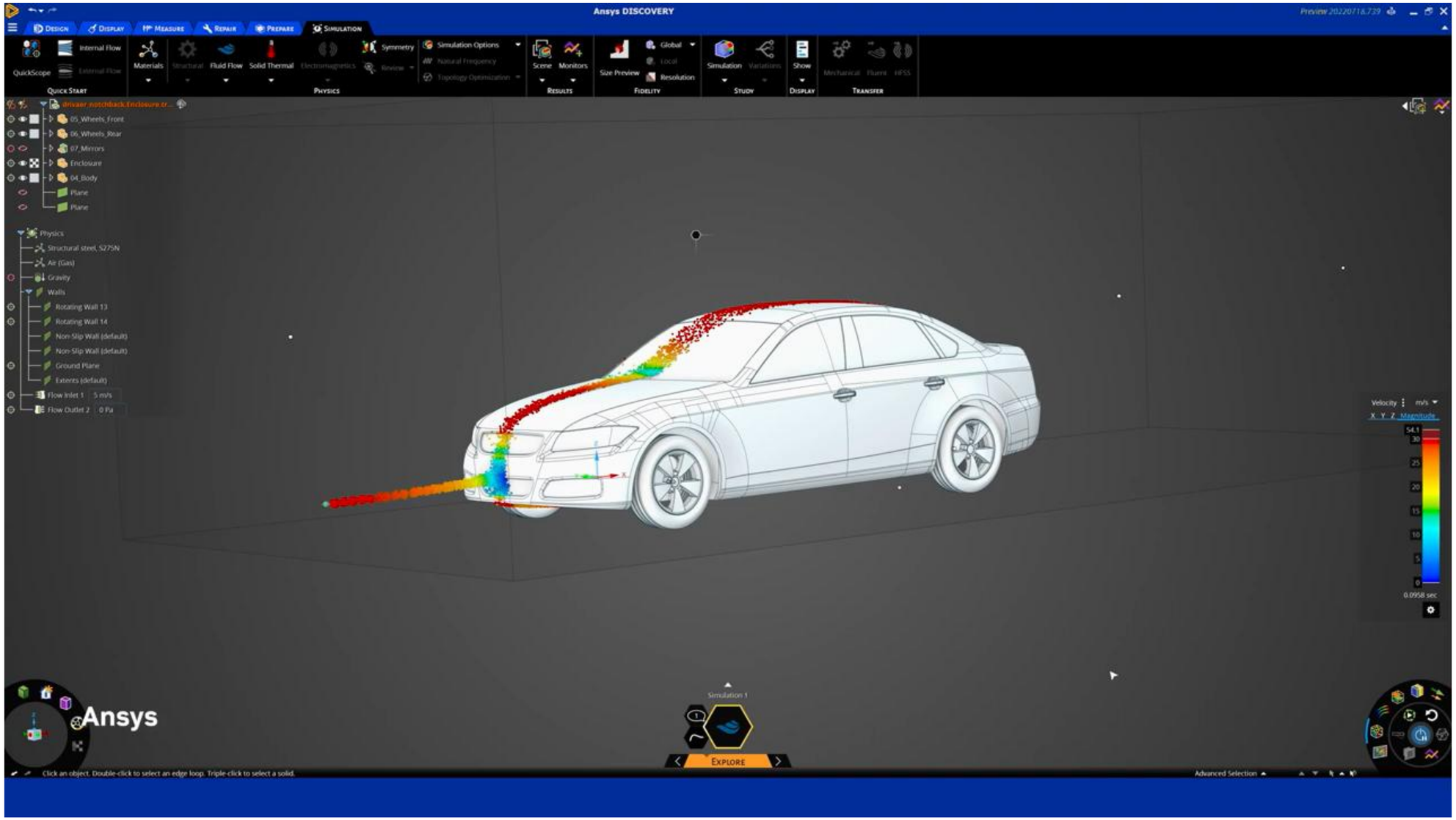
Largest and most complex climate simulation:
146 days per day at 1km resolution

JÜLICH
FORSCHUNGSZENTRUM

EuroHPC
Joint Undertaking

NVIDIA

Creating Real-Time Computer-aided Engineering Digital Twins with NVIDIA Omniverse Blueprints



How Does AI Help Identify useful Chemicals and Materials?

Traditional Chemicals Discovery

Human Intuition

Traditional Computations

Testing required with Trial and Error

Years

AI-driven Chemicals Discovery

AI + Human Intuition

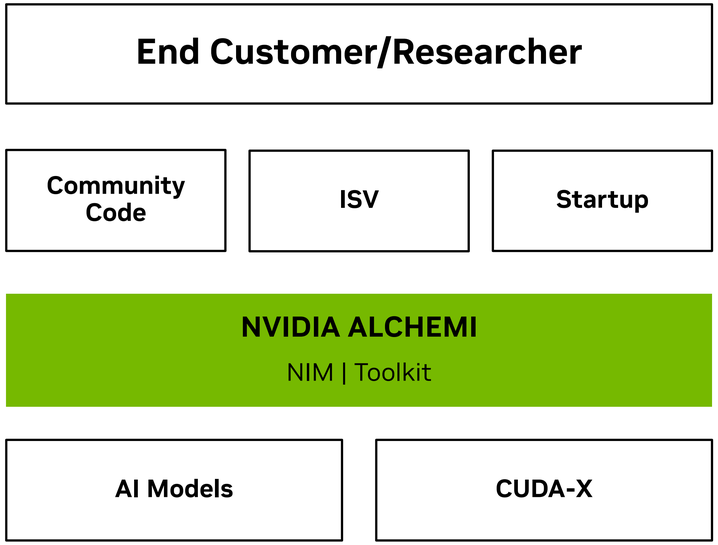
AI Computations

Testing required with AI-guided Search

Months

NVIDIA ALCHEMI

Usability, flexibility and performance



ALCHEMI Toolkit: **50k+ downloads** per month since Dec 2025 launch

GitHub: https://github.com/NVIDIA/nvalchemi-toolkit-ops

What is it

NVIDIA ALCHEMI (AI Lab for Chemistry and Materials Innovation) is a collection of domain-specific NIM microservices and Toolkit for accelerating chemical and materials discovery (e.g., battery, catalyst, OLED, beauty product):

•**NIM**: Scalable layer of cloud-ready microservices for chemistry and materials science, enabling deployment and orchestration on NVIDIA-accelerated platform

•**Toolkit**: Collection of GPU-accelerated batched kernels and tools for training Machine Learning Interatomic Potentials (MLIPs) and running accelerated batched atomistic simulations with MLIPs

Who is it for

- **Researchers**: Computational chemists and materials scientists in Industry and Academia requiring atomistic simulation at near quantum chemistry accuracy to inform chemical and materials discovery
- **Developers**: Community code, ISV[1] and AI-first startup developers training MLIPs and building accelerated batched MLIP atomistic simulation engines

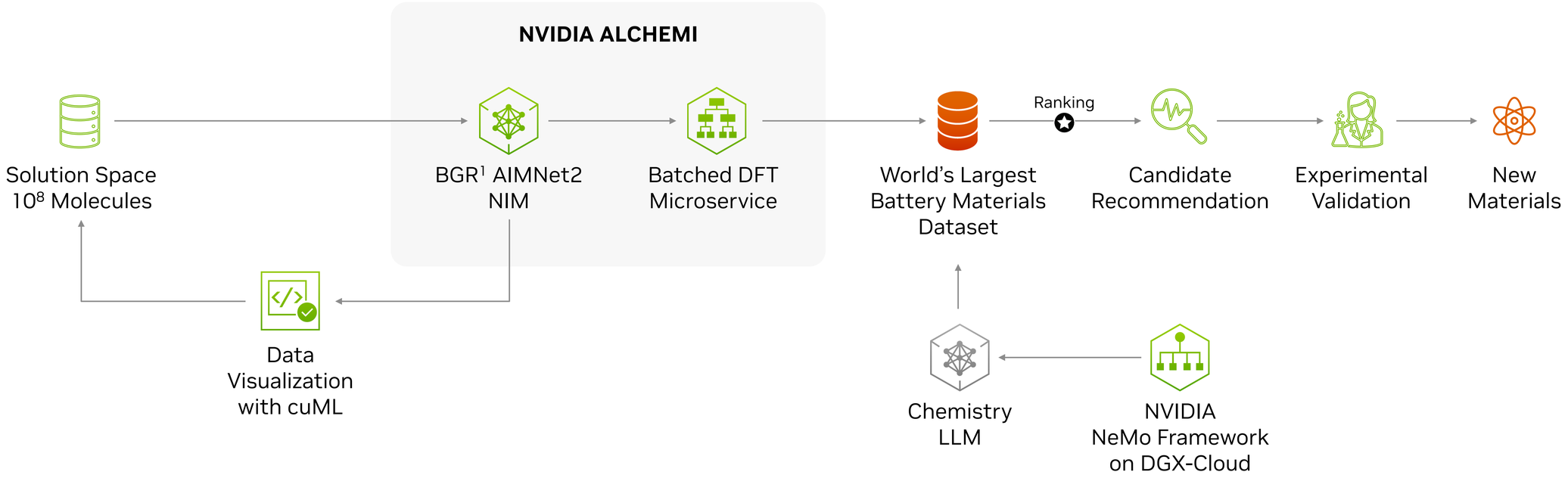
Value proposition

Accelerate chemical and materials discovery by enabling accurate, high performance atomistic simulations with simulation and AI and increasing successful outcomes in lab

1. Independent software vendor

AI-Accelerated Molecular and Materials Discovery

SES AI created world's largest battery materials dataset with NVIDIA ALCHEMI

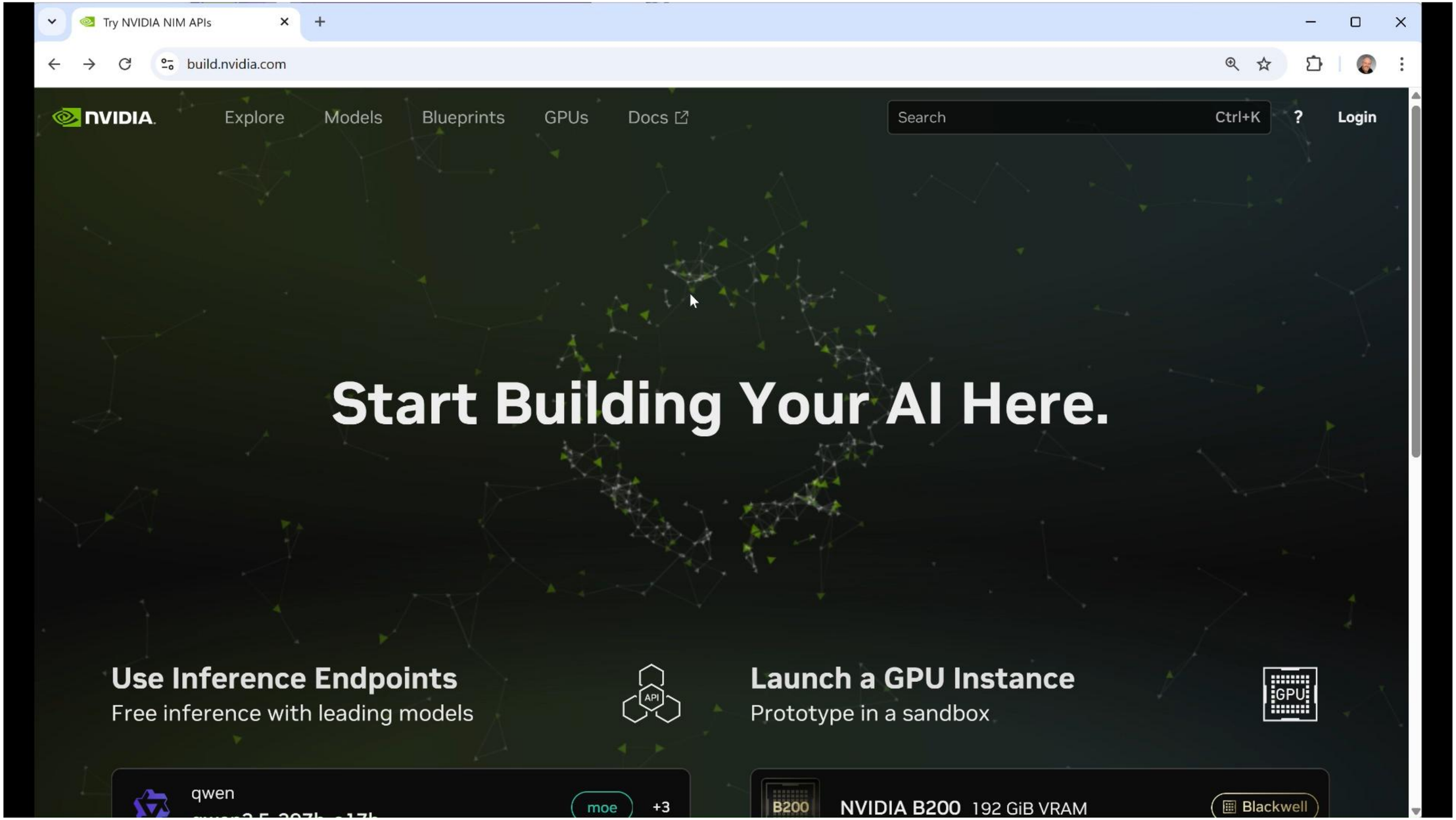


1,600x Acceleration[2]
(20 years to 4 days)

1. Batched Geometry Relaxation
2. 80x acceleration with NVIDIA ALCHEMI software and 20x acceleration from moving workloads to 500 H100s from CPU equivalence

NVIDIA

https://build.nvidia.com/



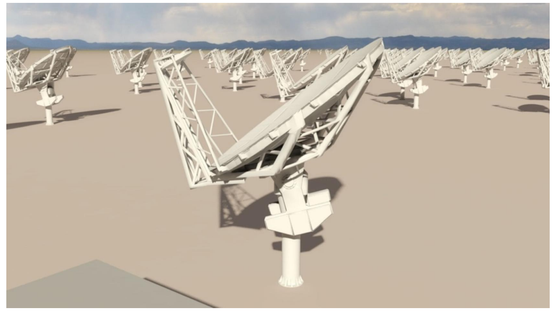
Next Generation Instruments are Producing Overwhelming Amounts of Data

Complexity of Experiments is Booming and Human Insight is Now the Bottleneck

Radio Astronomy

ngVLA – 244 Dishes

100 Petabytes per Year



SKA – 200 Dishes

1 Exabyte per Year

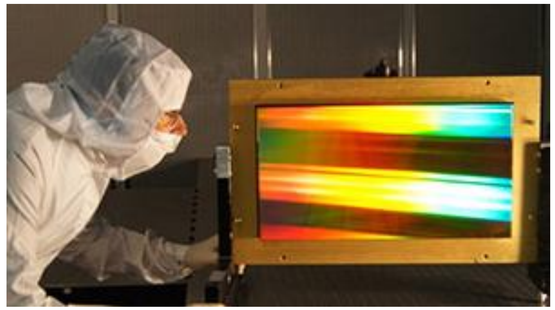
Particle Physics

High Luminosity LHC

100x Data than Higgs Boson Discovery

Advanced Laser Systems

100x Increase in Repetition Rate



Light Sources

APS-U – >60 beamlines

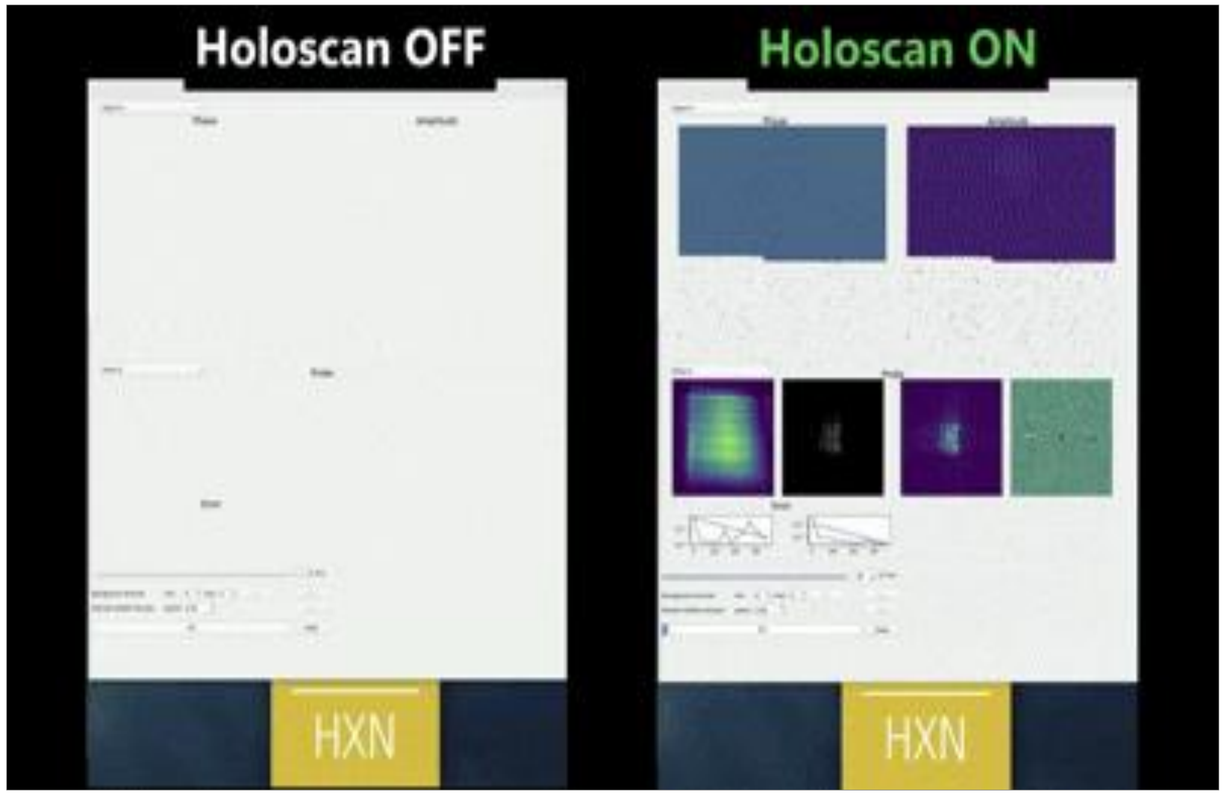
100-200 Petabytes per Year

Free Electron Laser LCLS-II

1 MHz Rep-Rate Upgrade

Real Time Nanoscale Imaging with NVIDIA Holoscan

Brookhaven National Laboratory NSLS-II, DECTRIS



Demonstration of Offline versus Online Ptychographic Imaging

Self-assembly from nanoparticles, potentially useful as optical and mechanical metamaterials

For visualization purposes, both offline and online videos have been sped up by 2x

- **Current beamline scan takes about 8 seconds, with reconstruction taking 1-2 minutes**
  - Requires full dataset from x-ray detector and disk storage prior to reconstruction
- **Impact of Edge HPC and Holoscan for real time imaging**
  - Real time reconstruction provides instantaneous user feedback on experiment results
  - Ability to handle higher data rates yields higher resolution experiments and improved science products
  - Laying foundation for AI-driven agents and autonomous experimentation, including AI based scan patterns, leading to the **discovery of new materials at unprecedented rates**

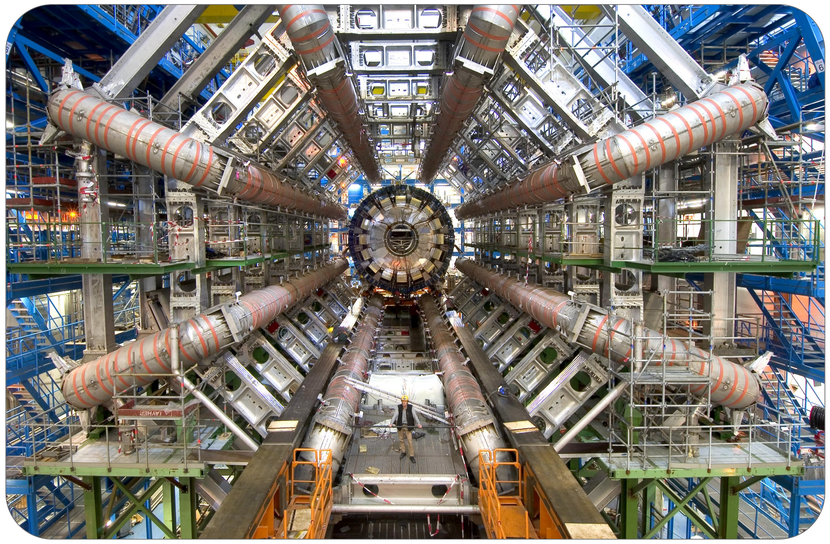
Brookhaven National Laboratory DECTRIS UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

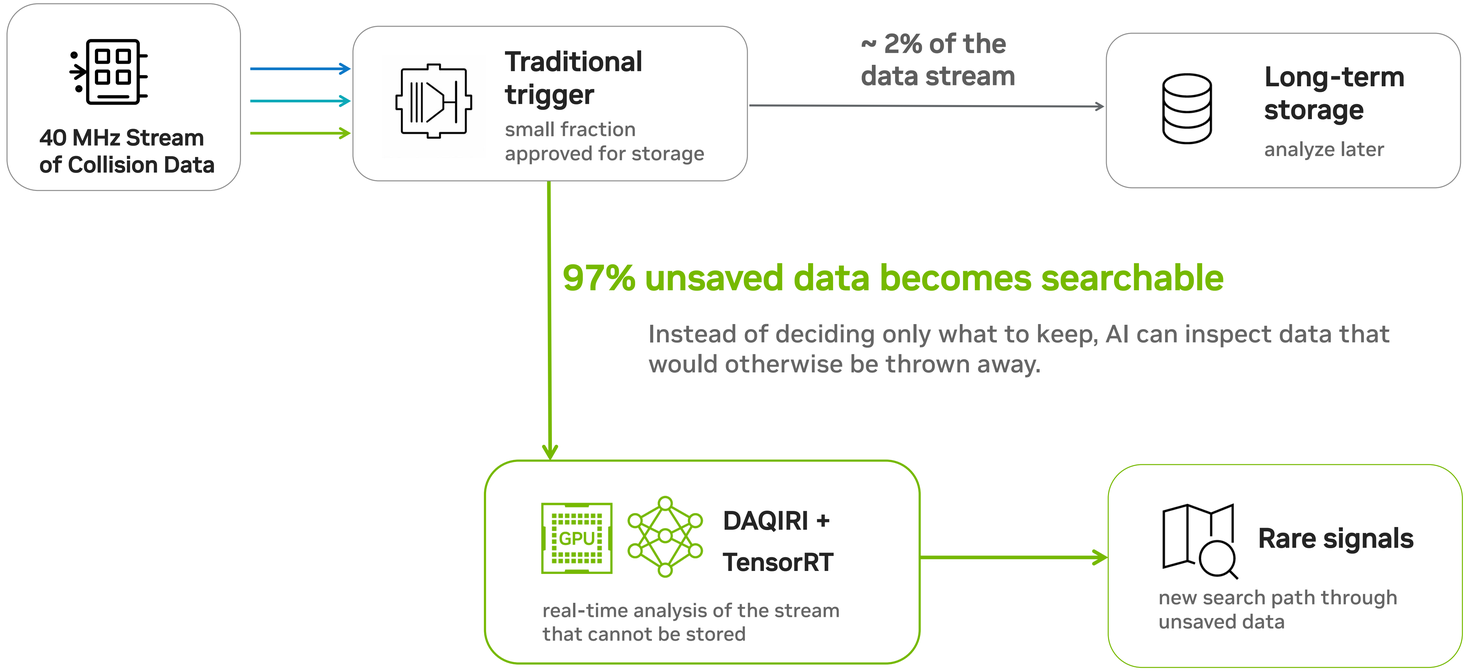
Qian Chen group, Department of Materials Science and Engineering, University of Illinois, Urbana-Champaign

NVIDIA

CERN A-GHOST: DAQIRI Enables Search In Data That Can’t Be Saved

High Luminosity Update raise the ATLAS data rates beyond the point viability of “Store First, Analyze Later”





97%

data becomes searchable

Allows Researchers to Search for New Physics in data stream

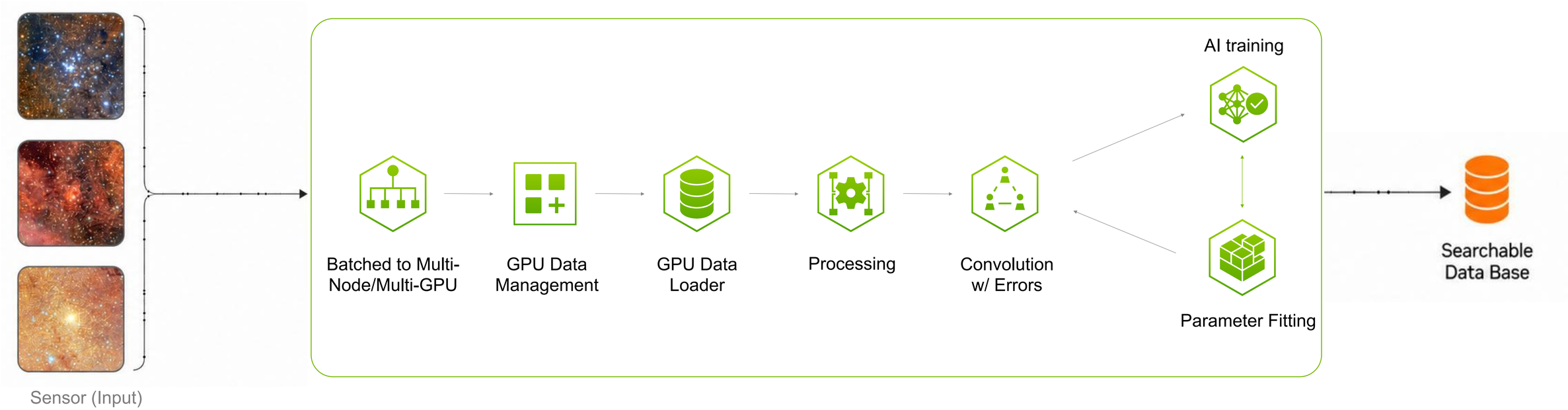
A-GHOST turns that unsaved stream into an AI-searchable path.

NVIDIA

NVIDIA cuPhoton

New library for telescopes, X-rays and laser experiments

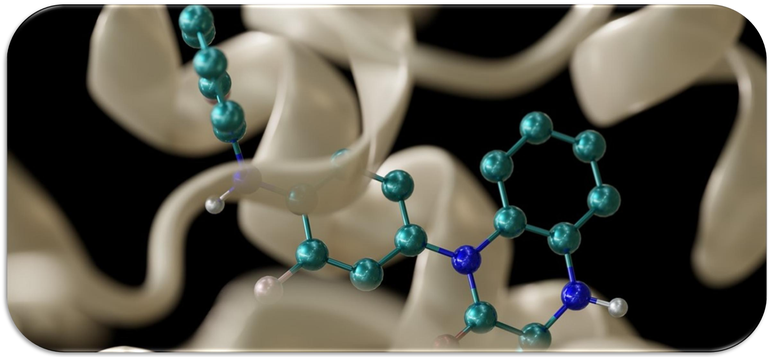
Enables analysis of full camera image in real time for scientific discoveries



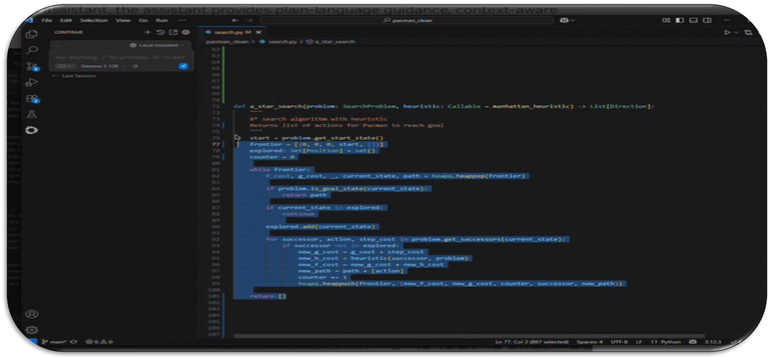
Accelerates image loading/reading by up to 15,000x, and processing and analysis by up to 8,000x

New Trend: AI Scientists

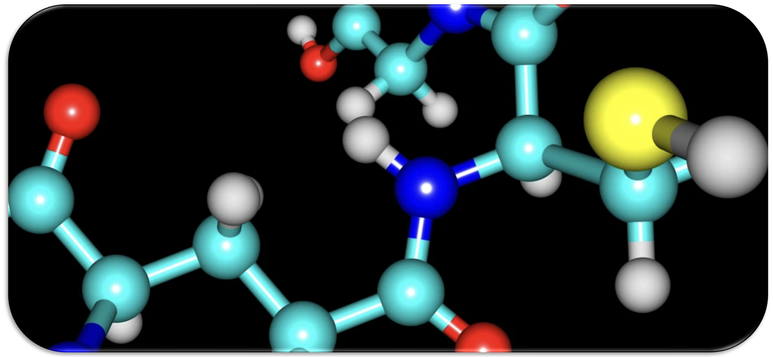
AI Scientists emerging across every domain



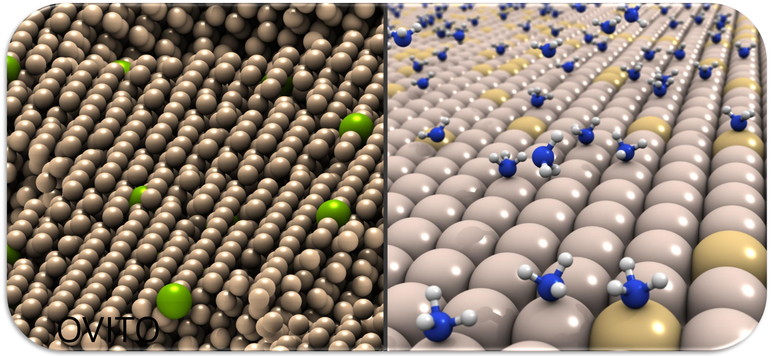
Automating Drug Discovery



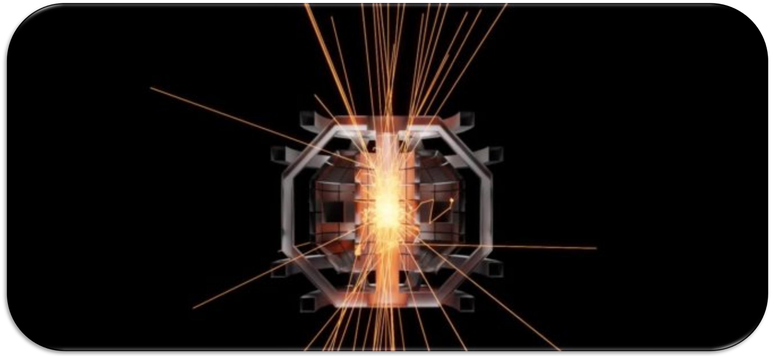
Automating Coding



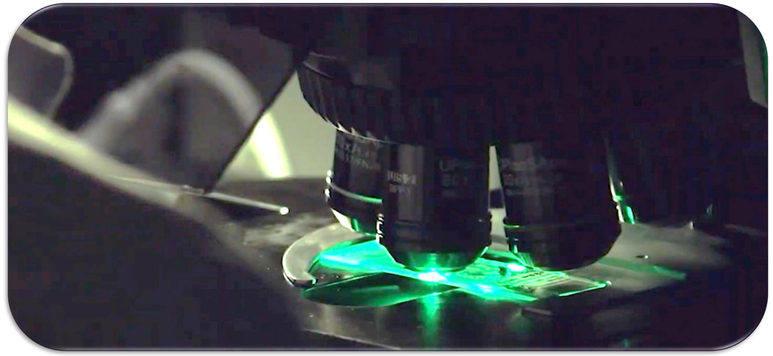
Designing New Molecules



Materials Candidate Generation

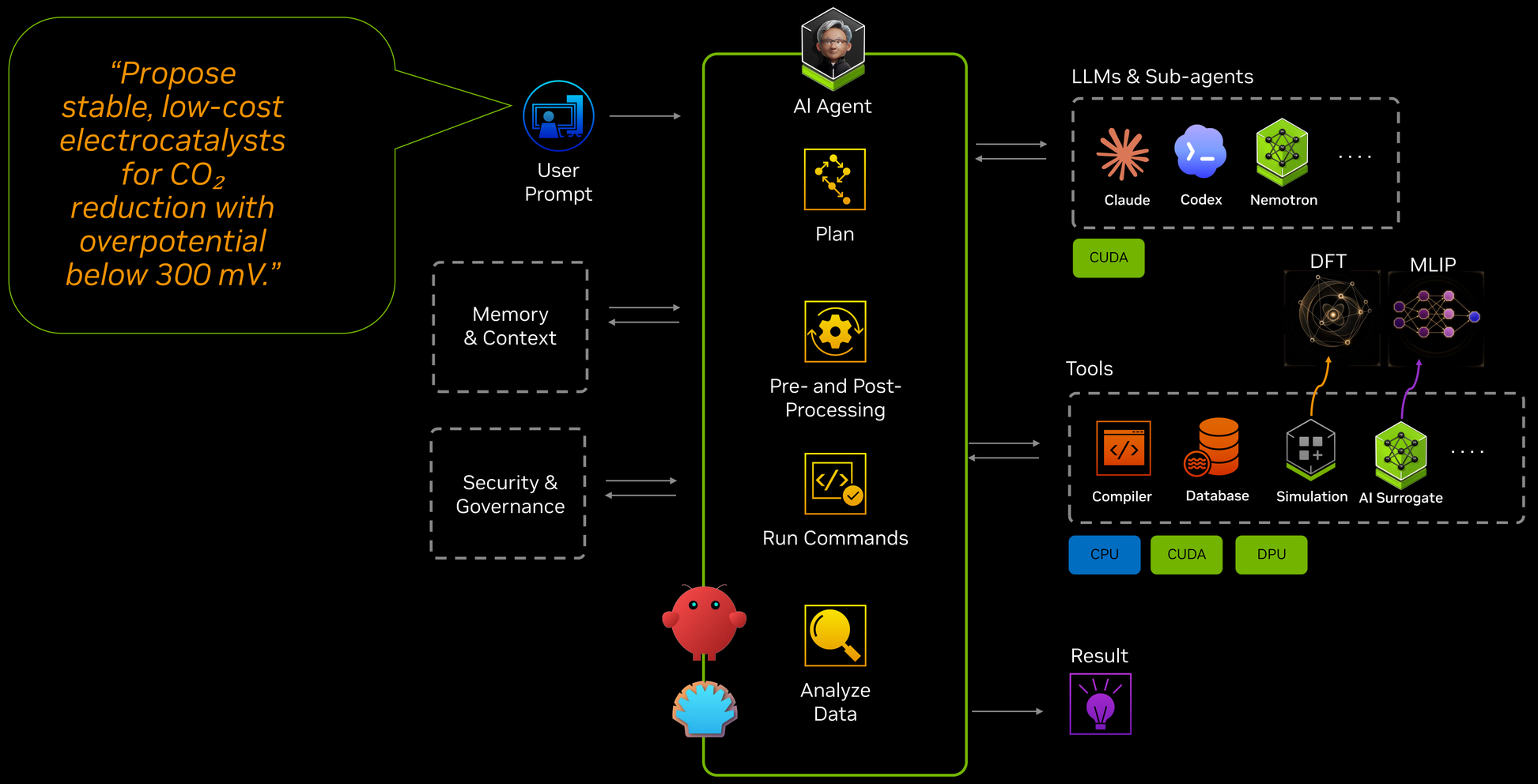


Fusion Hypothesis Generation

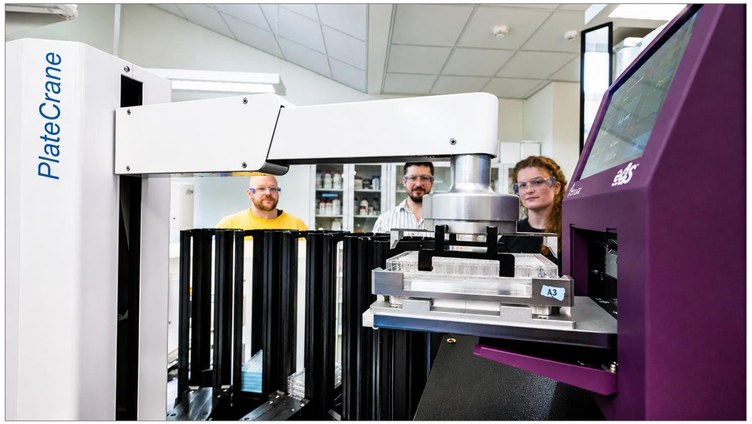


Optimizing Clinical Workflows

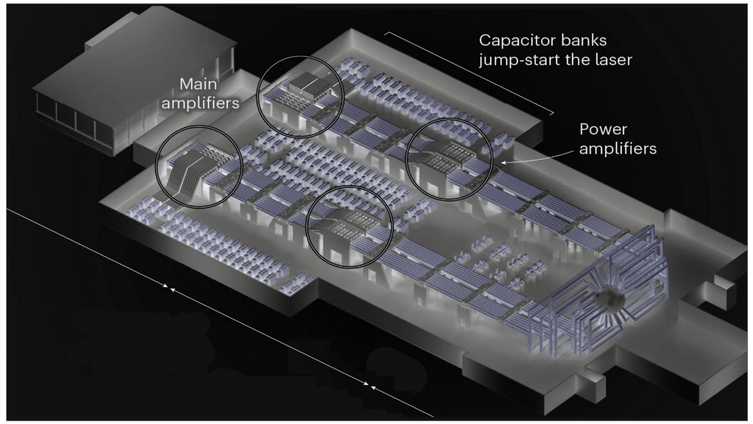
Agentic AI for Science



Agentic AI is Driving the Next Wave of Science and Engineering



ROSA: Robotic Scientific Assistant
Argonne National Lab



URSA: Universal Research Scientific Agent
Los Almos National Lab

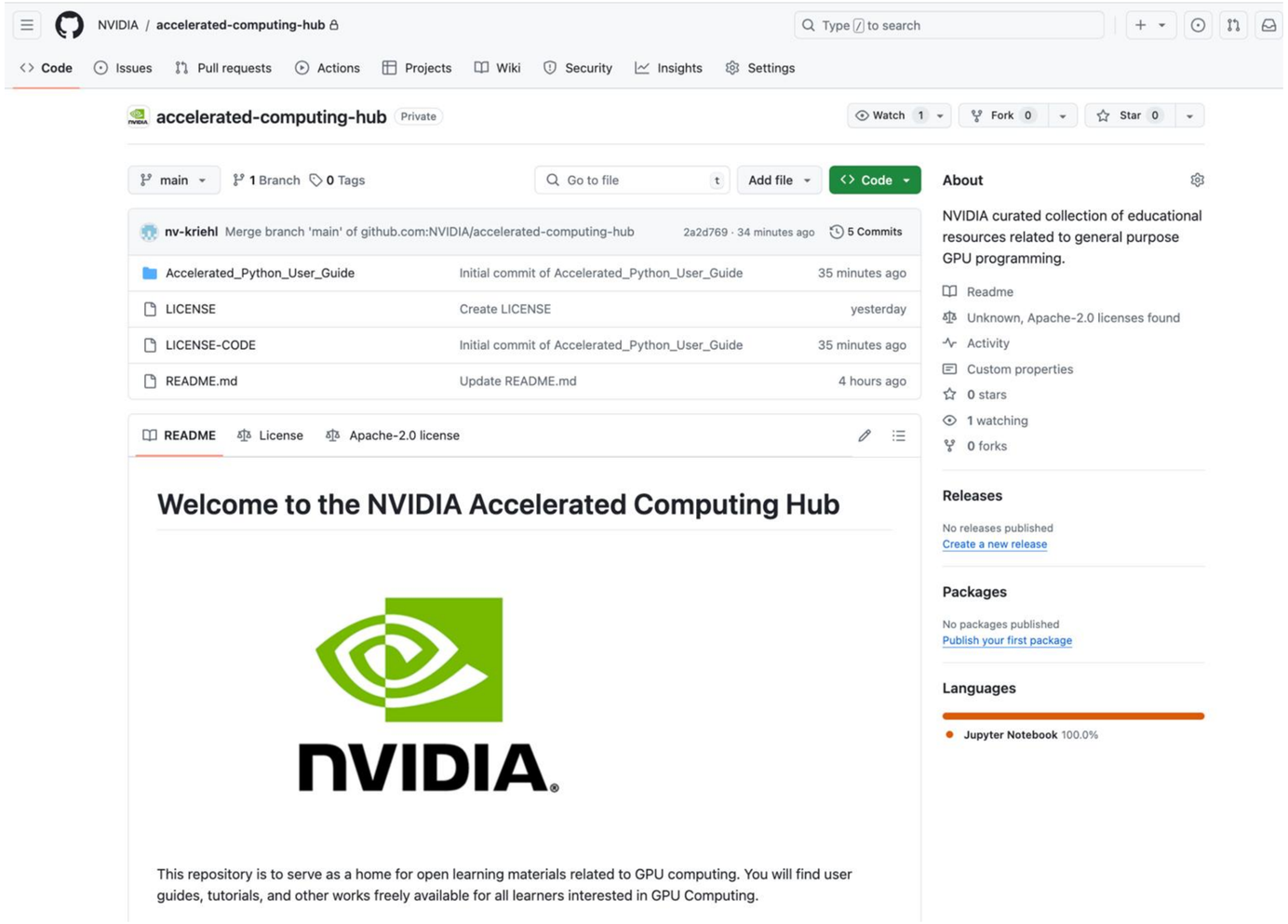
Automated Quantum Calibration
Aegiq

20 Years of CUDA: Honoring the Architects of the Accelerated Age

Accelerated Computing Hub

Open source resource for learning CUDA

Many examples, samples, tutorials available to get you started



https://github.com/NVIDIA/accelerated-computing-hub

GTC Is Coming to Berlin

Join NVIDIA GTC in Berlin - where developers, researchers, and business leaders will explore technologies transforming every industry.

It all starts with CEO Jensen Huang's keynote at the iconic Tempodrom, followed by sessions, hands-on labs, and networking.

Registration is now open.

Workshops October 20, 2026
Conference and Expo October 21–22

nvidia.com/en-eu/gtc

Thankyou!

NVIDIA
Accelerating Research

Paul Graham, Senior Solutions Architect

pgraham@nvidia.com

NVIDIA