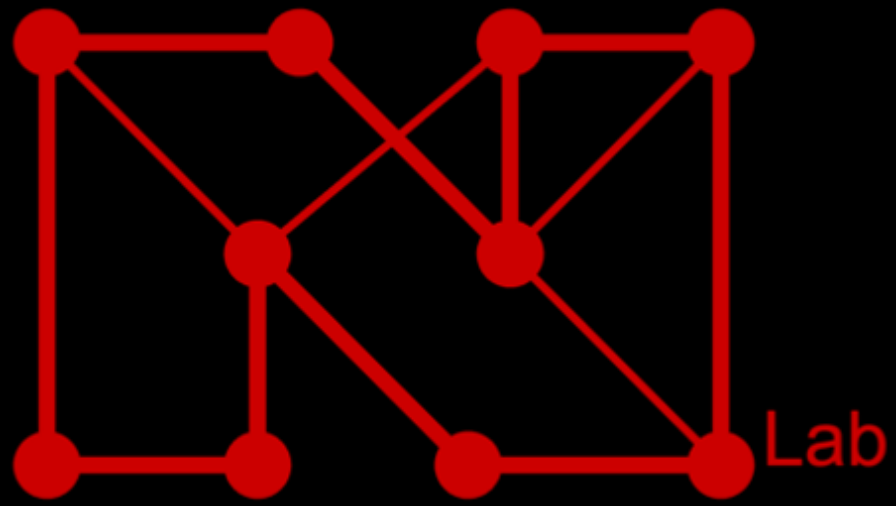
DRAFT

NNSIGHT AND NDIF: DEMOCRATIZING ACCESS TO OPEN-WEIGHT FOUNDATION MODEL INTERNALS

NEURAI Lab, Silicon Valley

Jose L Sampedro Mazon
sampedromazon.j@northeastern.edu



*"**AI systems are grown more than they are built**—their internal mechanisms are "emergent" rather than directly designed (…) **we have no idea, at a specific or precise level, why it makes the choices it does**, or why it occasionally makes a mistake despite usually being accurate.*

(…)

***I am very concerned about deploying such systems without a better handle on interpretability**. These systems will be absolutely central to the economy, technology, and national security, and will be capable of so much autonomy that **I consider it basically unacceptable for humanity to be totally ignorant of how they work**."*

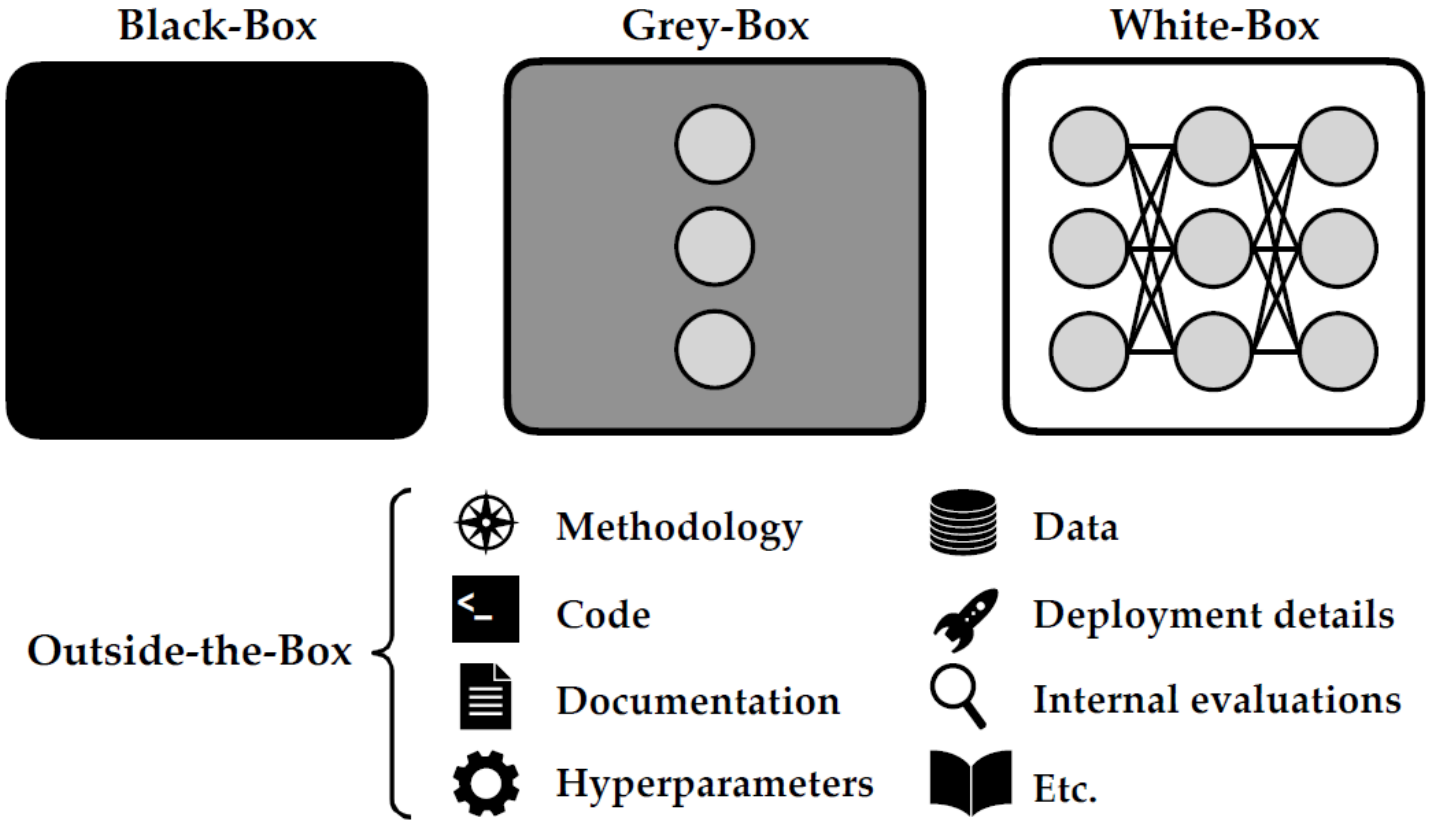
Source(s): https://www.darioamodei.com/post/the-urgency-of-interpretability

AI models offer different levels of access

Black-box access allows users to design inputs for a system, query it and analyze the *outputs (e.g. OpenAI ChapGPT / completions API).*

Grey-box models allow access to intentionally exposed model internals like inner neuron activations, input embeddings, probabilities, etc. (e.g. Cohere)

'De facto' white-box access allows running custom tests programmatically via API but system's parameters cannot be copied (e.g. PyTorch Hub models)



White-box access allows users full access to weights, activations, gradients, and enable model fine-tuning *(e.g. Llama 3, Falcon)*

Outside-the-box access can include methodological details, source code, documentation, hyper-parameters, training data, deployment details, and findings from internal evaluations *(e.g. Hugging Face Model Cards)*

Source(s): Stephen Casper, Carson Ezell et al.: Black-box access is insufficient for Rigorous AI Audits

Level of access dictates depth of interpretability studies

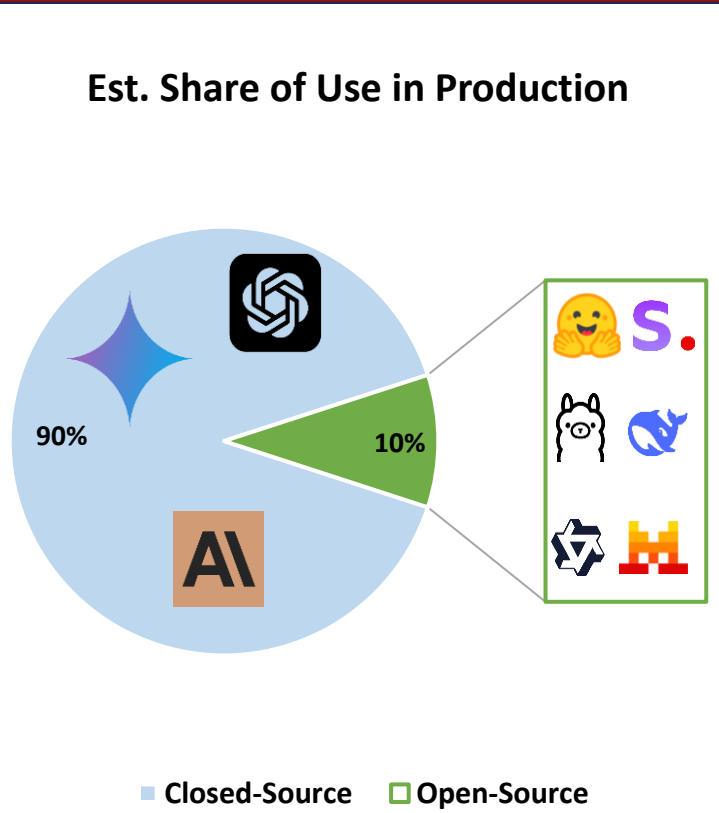
Access Type	Interpretability Potential	Key Limitations
Black-Box *(Prompt system and observe output only)*	✓ Test inputs & outputs ✓ Assess basic performance ✓ Identify obvious biases	✗ Hard detect rare failures ✗ Unable to identify problem sources ✗ Easy for developers to "game"
Grey-Box *(Limited API access to internals developers choose to expose)*	✓ Access predefined outputs only (logits, embeddings) ✓ Analyze basic confidence scores / probabilities ✓ Limited testing of explicitly exposed features	✗ Limited insight into internal representations ✗ Exact problem mechanisms may remain hidden ✗ Unable to modify or fine-tune model
'De facto' White-Box *(Broad API access to internals but cannot copy / extract full model parameters)*	✓ Access to internal activations at specific layers ✓ Run custom functions on exposed states ✓ Limited interventions (exposed activations) ✓ Maintain protection of model IP	✗ Higher latency for multi-step analyses / iterative experiments due to API calls ✗ Still restricted to approved intervention points ✗ No visibility over weights / attention patterns ✗ Limited causal tracing across model components
White-Box *(full access incl. weights, activations, gradients, fine-tuning)*	✓ Access model internals loading model locally ✓ Trace behaviors to specific components ✓ Reveal hidden capabilities via fine-tuning ✓ Complete causal analysis of behavior	✗ No insight into training methods or data ✗ Requires model-specific architectural knowledge ✗ Demands infrastructure expertise for large models
Outside-the-Box *(access to training data and deployment information)*	✓ Comprehensive inspection of model ✓ Full intervention at any computational point ✓ Evaluate development practices	✗ Cannot show how model processes information ✗ Documentation may be incomplete ✗ Requires model access for complete picture

Source(s): Stephen Casper, Carson Ezell et al.: Black-box access is insufficient for Rigorous AI Audits

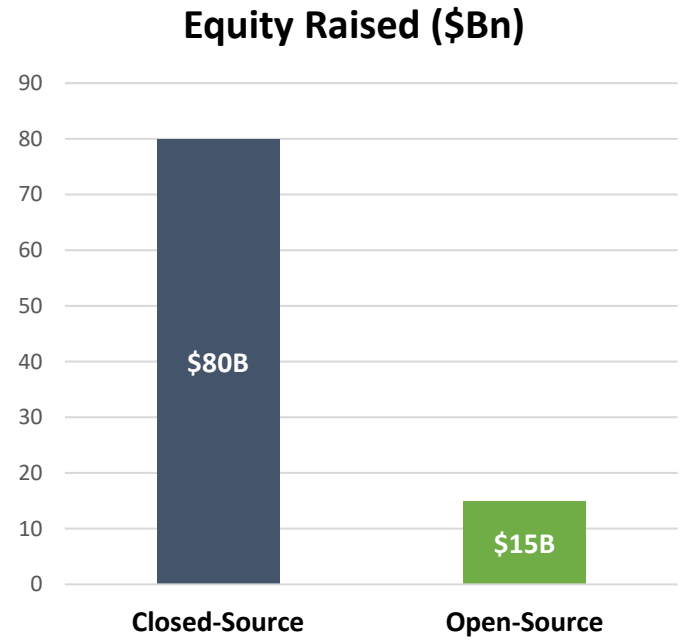
Black-box models dominate, limiting interpretability efforts

Priority to understand most widely used and capable models, but these are predominantly black-boxed

Adoption: Use of Closed-source models predominates with ~90% market share

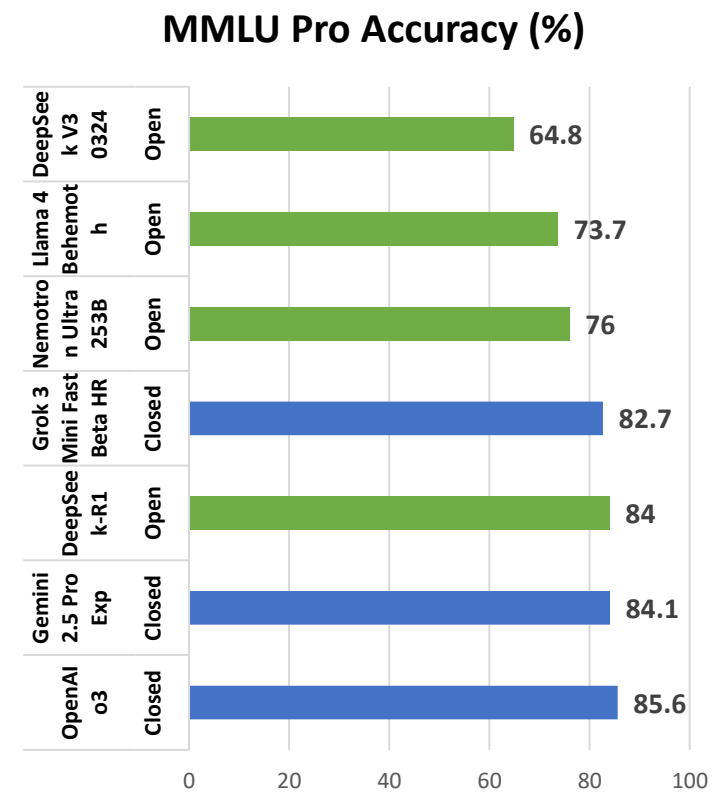


Investment: Closed-source developers attracted ~5x more funding since 2020



* Big tech open-source models (e.g., Meta's Llama) are internally funded and not included in these totals, but their share is still much smaller than the closed-source giants.

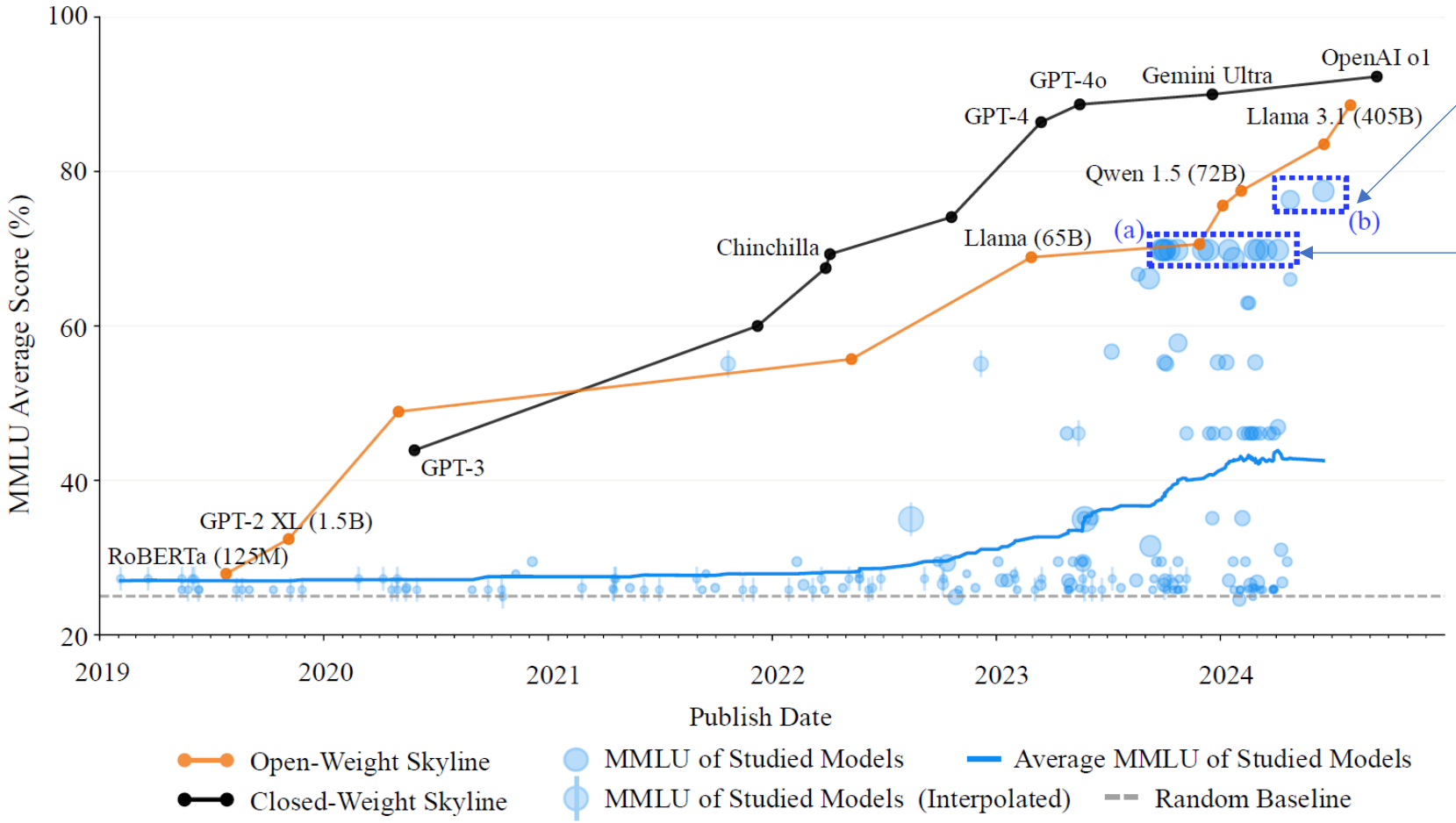
Capability: Closed-source still leads in performance but gap is closing rapidly



Source(s): CBInsights: The Foundation Model Divide (Jan 2025); GAI Insights: Estimated Market Share of Closed-Source LLM Models (Aug 2024); Vals AI MMLU Pro Benchmark (May 2025); Statista MMLU-Pro Trends (May 2025); Vellum AI Open LLM Leaderboard (Apr 2025)

Even for open models, studies of leading models are lacking

Most interpretability research is done on models that lag far behind SoTA capabilities.
This gap is extended even further when considering leading closed-weight model the performance.



(b) Only a small group of papers study language models with ≥70% MMLU performance

(a) **60.6%** of the surveyed research papers since Feb 2023 are still doing research on smaller, less performant models (< 40% MMLU) far behind the leading open-weight models (>405B)

Source(s): Jaden Fiotto-Kaufman, Alexander R. Loftus - NNSIGHT and NDIF: Democratizing Access To Open-weight Foundation Model Internals

Gap suggests engineering & infrastructural barriers to research

Interpretability research on >70B models is a leap in technical complexity & resource requirements

H100 GPUs	4	→	**~6x**	→	24
Total VRAM	280 GB	→	**~5.7x**	→	1.6 TB
Inference Cost	$0.80/M tokens	→	**15x**	→	$12.00/M tokens
Power	~2 KW (4 H100 GPUs)	→	**~8x**	→	~16 KW (24 H100 GPUs)
Network	100 Gbps (single node)	→	**~4x**	→	400 Gbps (multi-node)
Technical Complexity	Single-node tensor Stable at FP4 quantization Most MI tools support single-node		→		Multi-node pipeline + tensor parallelism Custom kernel optimizations Multi-TB Checkpoint Management Debugging with Distributed logs (takes days) High Error rates with quantization High downtime due to GPU / network failures

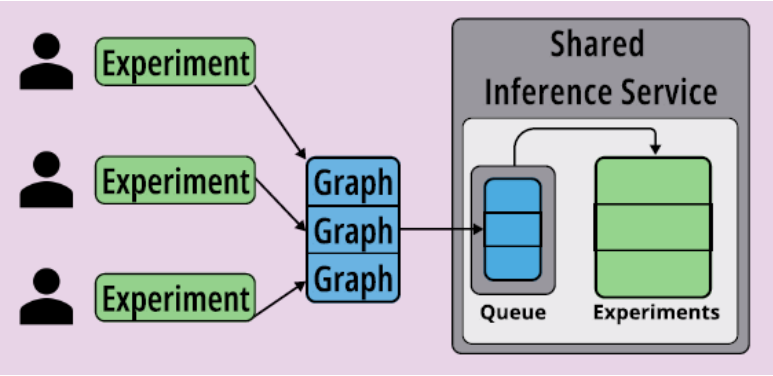
Note: these estimates reflect significant incremental costs involved in mechanistic interpretability experiments (VRAM overhead from activations and gradients, increased token processing, unviable quantization, etc.)

Source(s): Llama Official docs, Snowflake AI Research and Engineering Blogs, Snowflake, Sam Stoelinga blog (Google), reddit community benchmarks.

NDIF tackles infra barriers and complexity to study of large NNs

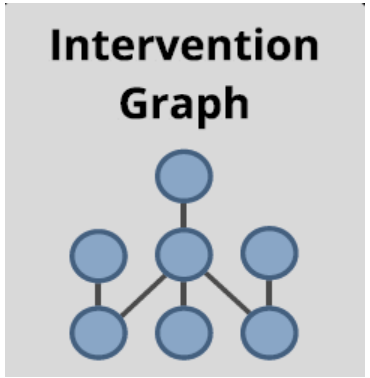
Co-tenanted compute, a new framework to design experiments and a data structure to represent them

NDIF Shared Inference Service minimizes infrastructural barriers



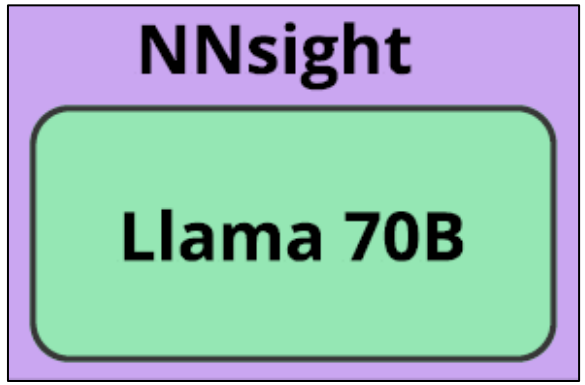
- Multi-user runtime infrastructure designed to ***amortize costs and reduce engineering burden***
- Designed to allow multiple researchers to conduct custom experiments while ***sharing the underlying pre-loaded model instances on shared compute*** HW

Intervention Graph structure separates experiment code to run on shared infra



- Data structure holding experiment code, ***dynamically captures data from and runs modifications to the model***. Stored in JSON, and sent to or retrieved from NDIF inference server for execution
- Decouples experimental and engineering code so that the ***complexity of running the model can be tackled separately***

NNSight Framework to define intervention graph and run remotely



- Python library that extends PyTorch to ***define and execution intervention graphs***, introducing deferred remote execution.
- Trace context window provides access to the intermediate inputs and outputs of all modules
- Allows the user to express all the experiments ***as if using PyTorch locally***.

NNSight extends PyTorch for intuitive, transparent interventions

Crucially, NNSight implements and manages intervention graphs at run time between remote & local

Intuitive API Design	• Extends PyTorch with **familiar syntax** for interventions • Uses **context manager** (with *model.trace()*) for defining experiments • **No need to manually register/remove hooks** for each experiment *(Standard PyTorch requires creating custom hooks for each access point)*
Deferred Remote Execution	• Operations ***recorded to intervention graph for later remote execution*** within single forward pass • Creates ***proxy objects representing tensor values to be captured*** for later interaction • Marks to ***preserve tensor values after execution*** with *.save()* • *FakeTensor* Integration enables ***experiment debugging without full execution***
Extensive direct access & interventions	• ***Access*** any model inputs, outputs, and gradients • Support for ***complex interventions*** (e.g., activation patching, causal tracing) • Provides consistent ***access to both activation values*** (forward pass) ***and gradients*** (backward pass) • ***Compatible*** with a wide range of built-in and custom PyTorch model architectures
Local-Remote Flexibility	• ***Same code runs locally*** or on NDIF servers by simply adding *remote=True* • ***Debug*** on small models locally, then run identical code on larger models to debug effectively • ***No distributed programming needed*** - identical API abstracts away model sharding and parallelism

Neural networks can be represented as computational graphs

Computational graph as an alternative representation of NNs to the traditional ‘neuron’ model

Traditional ‘Neuron’ Representation blends both data and computation into discrete ’neuron’ layers

The traditional neuron concept is a simplified teaching model that combines:

- Parameters (weights & biases)
- Computation (weighted sum & activation)
- Data flow (inputs & outputs)

A "neuron" is described as if it:

- Receives inputs
- Computes a weighted sum
- Applies an activation function
- Sends an output forward

Computational Graph Representation separates these more accurately reflecting how NNs are implemented

There are no "neurons" as unified entities, but tensors and matrix operations (closer to PyTorch):

- **Variable Nodes (Tensors):** Weight matrices by bias vectors by layer; input (from previous layer) & output (to next layer) activation tensors
- **Apply Nodes (Operations):** Matrix multiplication (input × weights), bias addition, activation function application

Weakly connected, bipartite, acyclic graph with directed edges linking variable and apply nodes:

- Input Tensor (Variable) → Matrix Multiply (Apply) → Intermediate Tensor (Variable) → Activation Function (Apply) → Output Tensor (Variable)

Intervention graph is a structured way to modify computational flow

Interventions are modifications to the computational graph that interleave new nodes and edges

An intervention graph modifies a neural network's computational flow by:

1. **Creating a secondary computation graph** (C') with custom operations
2. **Connecting it to the original graph** (C) at specified points
3. **Executing the combined graph** in a single forward pass

The secondary computation graph (C') is comprised of two types of nodes (with corresponding edges):

- **Getters (access hidden states)**: Extract tensors from original computation, connecting *variable nodes* in *original graph* (C) to *apply nodes* in *intervention graph*
- **Setters (replace hidden states with modified values)**: Insert modified tensors back into the original computation, connecting *variable nodes* in *intervention graph* to *apply nodes* in *original graph* (C)

```python
from nnsight import LanguageModel
model_id = "meta-llama/Meta-Llama-3.1-8B"
lm = LanguageModel(model_id)


mlp = lm.model.layers[16].mlp.down_proj
neurons = [394, 5490, 8929]
prompt = "The truth is the"


with lm.trace(prompt, remote=True):
    mlp.input[:, -1, neurons] = 10
    out = lm.output.save()


last = out["logits"][:, -1].argmax()
prediction = lm.tokenizer.decode(last)
print(prediction)
```

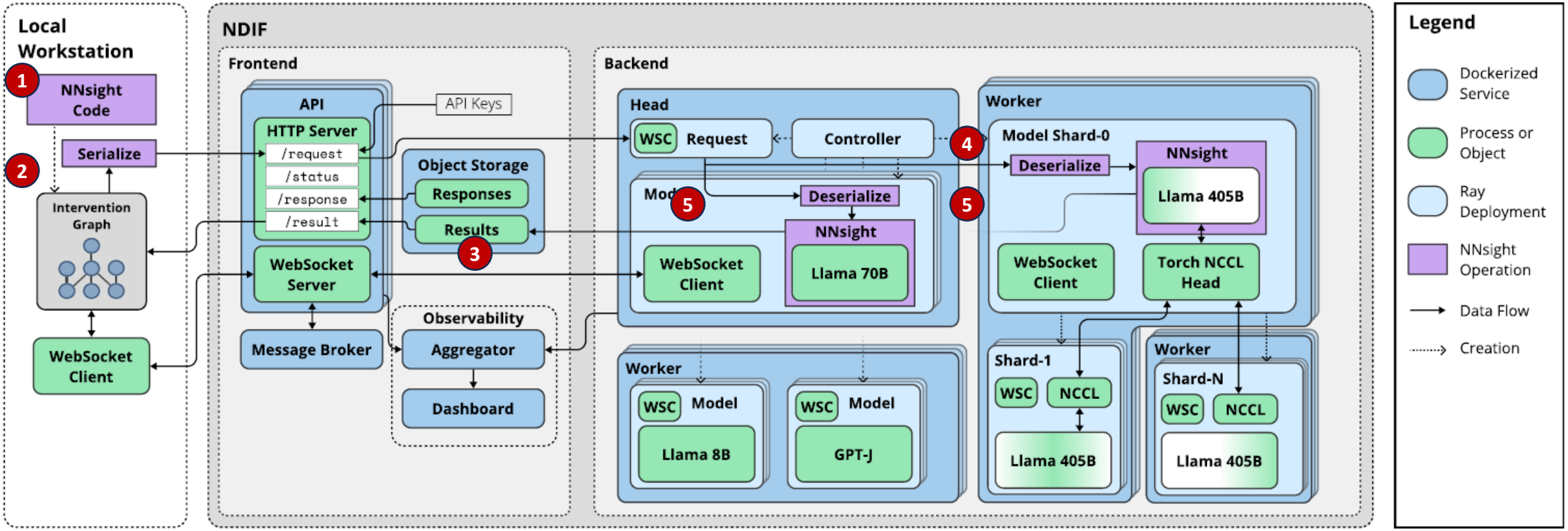
Getter: accesses the raw tensor values from specified variable node in layer 16 of the NN

Intervention Operation: Sets specified tensor elements controlling "sematic inversion" to a high activation value and save result to 'out'

Setter: Modified values are implicitly inserted back *(NNsight handles this automatically when a tensor is modified)*

The most probable token predicted – after the intervention operation - is extracted and decoded to English, resulting in "The truth is the *lie*"

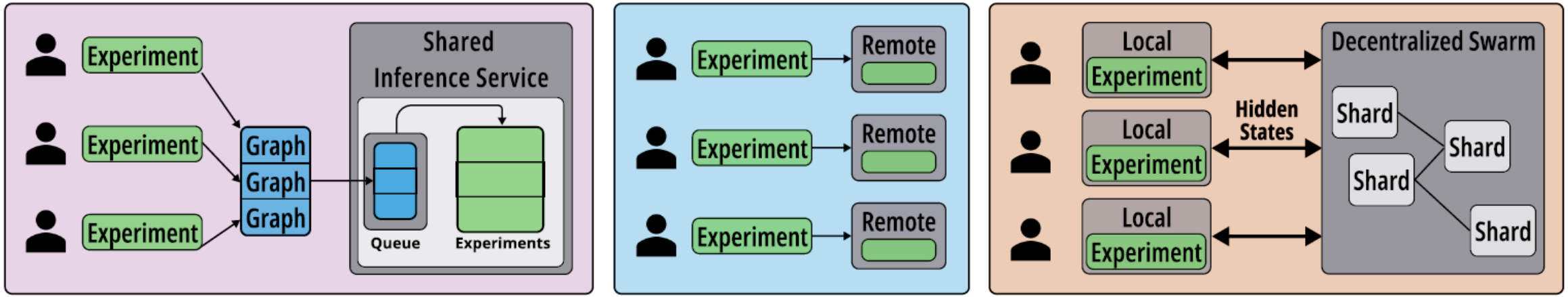
NNsight + NDIF work in tandem to enable interpretability studies



1. Researchers write experiment code using the NNsight API
2. Experiment code is converted to an Intervention Graph and sent to the NDIF frontend server
3. The intervention graph is routed to a worker node at NDIF's backend, which hosts multiple models
4. For larger models, (e.g. Llama 3.1 405B), model weights are distributed across shards (tensor parallelism)
5. Results are then gathered and sent to the object store in the NDIF frontend, then pulled to local workstation

* The graph can either be executed locally, or it can be sent to the remote NDIF servers for execution

NDIF vs. Traditional Approaches: Eliminating Infra Barriers



NDIF

- Multiple researchers share model instances
- Models are pre-loaded, no startup delay
- Experiment code separate from model infra
- Costs amortized across many users
- Abstracts model parallelism complexity
- Computations execute entirely on server
- Simple API with local/remote compatibility

HPC

- Separate model instance per researcher
- Extensive model loading times per session
- Researcher manages experiment + infra
- Full GPU cluster cost per researcher
- Requires expertise in distributed systems
- Computations stay on researcher's VM
- Complex distributed programming

PETALS

- Model distributed across volunteer nodes
- One-time loading, but network latency
- Researcher runs local experiment code
- Distributed costs (limited to largest models)
- Requires P2P networking knowledge
- Hidden states transfer downloaded locally
- Custom API with limitations on interventions

NDIF democratizes frontier model research by providing managed shared compute, eliminating the prohibitive costs of traditional HPC and the network bottlenecks of peer-to-peer approaches like Petals

NDIF as Industry Standard: Enabling ‘De Facto White-Box’ Access

At ~90% ‘market share’, it is an imperative for NDIF to seek adoption by SoTA proprietary model providers

“A key limitation is that [NDIF] does not provide access to closed-parameter, proprietary models like GPT-4 or Claude.

Proprietary vendors often (…) wish to maintain secrecy and security of model parameters for business and safety reasons. Our system is designed with these concerns in mind.

We encourage commercial providers to consider adopting our approach as part of their products to enable transparency and facilitate future scientific progress at the frontiers of AI”

Commercial Adoption Pathways

- **Establish NDIF as industry standard** for "de facto white-box" access while protecting proprietary models
- **Implement middleware solutions** that either embed NNSight or connect NDIF protocols to the existing commercial APIs
- **Enable selective transparency** through API extensions exposing specific activations without weight access
- **Bridge the gap** between fully closed systems and open-weight models to support transparency

Multi-Stakeholder Benefits

- **Preserve commercial IP** while demonstrating transparency commitment
- **Enable research of** frontier models powering 90% of deployed AI systems
- **Support regulatory compliance** with emerging AI transparency frameworks
- **Build user trust** through verifiable model inspection and understanding

Strategic Partnership Opportunities

- **Integrate with cloud providers** (AWS, GCP, Azure) as managed service
- **Connect with MLOps platforms / HuggingFace** to streamline adoption
- **Collaborate with hardware manufacturers** (NVIDIA) to optimize performance and embed NDIF/NNSight into model-agnostic toolkits
- **Partner with open model providers** (Meta) to expand ecosystem reach
- **Engage regulators, think tanks** and lobbies to become regulatory standard

Appendix

Black-box models severely limit interpretability studies

Access Type	Capabilities & Interpretability Goals	Limitations
Black-Box	• ***Test inputs/outputs*** to assess basic performance • Run ***manual red-teaming*** to find obvious flaws • Analyze ***response patterns*** to detect biases • Perform ***statistical analyses*** to measure fairness • Create ***simple behavioral tests*** to map boundaries	• Cannot detect ***rare or unexpected failure modes*** • Unable to identify ***which model parts*** cause failures • Findings ***limited by the inputs*** you think to test • ***Developers can optimize for test cases*** without fixing core issues • ***No way to verify*** if model ***explanations*** are truthful
Grey-Box	• Access ***token probabilities*** to analyze uncertainty • Examine ***basic activation patterns*** to trace influences • Use ***probability-guided attacks*** for more efficient testing • Analyze ***confidence scores*** to detect potential issues • See ***limited internal states*** to improve attribution	• ***Cannot pinpoint exact neural connections*** causing problems • Limited ability to ***uncover hidden capabilities*** • Unable to use ***efficient gradient-based*** testing methods • ***Cannot modify*** the model to test how easily safeguards break
De facto White-Box	• Run indirect processes via API for ***structured probing*** • Use ***transfer-based attacks*** to find vulnerabilities systematically • Perform ***gradient-free optimization*** for automated testing • Test ***model manipulation*** without direct weight access • Run ***batched experiments*** to detect internal biases	• Only ***indirect access through an API***, not to actual model code • Testing ***restricted*** to what the developer's interface allows • Some ***analyses take longer*** due to API limitations • Limitations on certain internal ***pattern analyses*** • ***Harder to trace specific behaviors*** to exact model components
White-Box	• Analyze weights to ***attribute behaviors to specific components*** • ***Fine-tune models*** to reveal hidden/dormant capabilities • Study ***internal representations*** to identify concept formation • Run ***gradient-based analyses*** for mechanistic understanding • Perform ***latent space attack***s to test safety boundaries thoroughly	• ***Requires specialized technical knowledge*** to analyze effectively • Analyzing large models needs ***substantial computing resources*** • No visibility into ***how the model was trained or on what data*** • Cannot determine ***why specific patterns were learned***
Outside-the-Box	• Review training methodology ***to trace problems to decisions*** • ***Analyze training data*** to assess bias and quality issues • Study documentation to ***evaluate development practices*** • ***Examine deployment s***ettings to understand real-world risks • Assess developer practices for ***accountability***	• Does not show ***how the model actually processes*** information • ***Documentation may contain gaps*** or inaccuracies • Hard to connect specific ***development choices to model behaviors*** • Companies may still ***withhold proprietary technical details*** • ***Cannot detect emergent properties*** without examining the model

NORTHEASTERN