Introduction to Neural Network Interpretability

Part II: Geometry, Circuits, Control, and Evidence

Contact Information
For permission requests or inquiries, please contact: ruchirtewari@gmail.com

Introduction to Neural Network Interpretability

Part II: Geometry, Circuits, Control, and Evidence

Chapters 5–8

Ruchir Tewari

September 7, 2026

How to Read This Book

Interpretability research moves quickly, but the questions underneath it are stable. This part of the book covers four of them, one per chapter:

1. **Chapter 5 - Representation Geometry.** What kind of mathematical object is a network's internal state? (Answer: points on low-dimensional manifolds, with concepts often laid out as directions, and more concepts packed in than there are dimensions.)
2. **Chapter 6 - Features, Circuits, and Causal Verification.** Can we name the parts and trace the wiring? (Answer: increasingly yes, via sparse autoencoders and circuit analysis - with important caveats about proof.)
3. **Chapter 7 - Control and Steering.** If concepts are directions, can we drive? (Answer: yes, surprisingly well for some behaviors, and the engineering trade-offs against prompting and fine-tuning are becoming quantifiable.)
4. **Chapter 8 - Evaluation and Frontiers.** How do we know an interpretation is *true*, and what should we work on next? (Answer: causal tests are the standard, faithfulness is the central unsolved problem, and the field's center of gravity is shifting to applied questions.)

These four chapters answer, in order, the four questions Part I posed at its close (its Section 3.5): what do the activation vectors represent, and how is it arranged? Which assemblies of components implement which behaviors? How can behavior be changed by editing the inside rather than retraining the whole? And when an explanation is offered, how do we know it is true of the machine rather than merely plausible to us? Part I's foundations - the statistical and geometric lenses (its Chapters 1–2), the machines (Chapter 3), and their training dynamics (Chapter 4) - are assumed throughout; where this part leans on them, it points back explicitly.

Some conventions used throughout:

- We always separate *correlational* evidence (a probe can decode a quantity from the activations) from *causal* evidence (intervening on the activations changes behavior). This distinction does most of the epistemic work in this field.
- Short **Neuroscience parallel** paragraphs appear inline where a method or question was inherited from systems neuroscience. Mechanistic interpretability borrowed several of its core methods - receptive-field mapping, lesioning, representational similarity analysis, population-level analysis - and knowing the lineage tells you which failure modes to expect.
- Citations appear inline as (Author, Year); full references, with the filenames of the archived PDFs, are collected in the References chapter at the end of the book.
- Chapters close with **Takeaways** (the chapter in a few sentences), **Research Directions** (open problems, scoped so a reader could start on them), and **Programs** (hands-on projects that let you reproduce the chapter's main claims yourself). They are placed at the end of each chapter so they do not interrupt the main text; readers looking for project ideas can go straight to those sections.

- Appendix A provides a year-by-year timeline of interpretability and model-biology research from 2019 to 2026, so the results discussed in the chapters can be placed in historical order.

All programs run on a laptop with Python 3, `numpy`, `matplotlib`, `torch`, and `transformers`, using GPT-2-small-class models.

Contents

5. Representation Geometry

When a transformer processes the sentence *"The Eiffel Tower is in"*, every token becomes a vector (768 numbers in GPT-2 small, over ten thousand in frontier-scale models) and every layer rewrites those vectors. Interpretability begins with a simple question: *what kind of object is that vector?* This chapter answers it in three moves: fix the vocabulary (spaces, maps, activations, representations), measure the shape of the activation cloud (manifolds, superposition, the structure found at scale), and then act on it - first by reading it through probes and lenses, then by applying pressure to it. Words the field uses loosely - *latent space*, *representation*, *feature*, *direction*, *manifold* - each get one meaning here and keep it for the rest of the book.

Structure of this chapter. Section 5.1 lays the foundation: the vector spaces the architecture fixes, the learned maps between them (four architectural facts with long shadows - the compressing embedding, the additive residual stream, the per-head rank bottleneck, depth as iterated refinement), and the activation vectors those maps actually produce on data. Section 5.2 defines **representation** and distinguishes it from the vector that carries it and from the space that contains it. Sections 5.3–5.5 study the geometry of the activation cloud itself: it concentrates on **low-dimensional manifolds** (5.3); it packs **more features than dimensions** into overlapping directions - superposition (5.4); and, measured at scale, it has reproducible shape - feature crystals and lobes, a pronounced anisotropy, partial universality across models - with the shape reorganizing as the input changes across languages, code, and formats (5.5). Section 5.6 turns to instruments: probes, the logit and tuned lenses, and affine stitching between layers and models - all of them *projections* from the latent space to something a human can check, and the home of the **linear representation hypothesis**. Section 5.7 turns from reading to pressure: the moves that work on a frozen latent space, and the training-time levers that shape what forms in it. Section 5.8 closes by distinguishing representations from **conceptors**, the region-summarizing operators that Chapter 7 uses for control. Open problems and hands-on programs are collected at the end.

5.1 The Spaces, the Maps, and the Points

5.1.1 The spaces: vectors, subspaces, directions

The raw objects come first, stripped of every learned association. A **vector** is a list of d numbers; the **vector space** $\mathbb{R}^d$ is the set of all such lists, closed under addition and scaling. Nothing about $\mathbb{R}^{768}$ is specific to GPT-2 small: it is an empty container whose width is fixed by the architecture, the same featureless space before and after training. Meaning will enter only through the maps (next subsection) and the data (the one after).

Three pieces of structure on this container do all the work in the chapter. A **subspace** is a flat slice through the origin - a line, a plane - closed under addition and scaling. A **direction** is a one-dimensional subspace with a chosen orientation. An **inner product** turns "how much of direction $\hat{v}$ is present in x" into a single number $\langle x, \hat{v} \rangle$ - the geometric primitive behind every probe, every steering vector, and every logit in this book.

Finally, a **basis** is a choice of coordinates. The neuron basis - one axis per unit - is *one* basis of the activation space, distinguished from all others only by the fact that the elementwise nonlinearity is applied along it. Nothing in the mathematics forces learned structure to align with it - a one-sentence piece of linear algebra that will defuse most of the surprise in Section 5.4's finding that single neurons respond to unrelated mixtures of things.

5.1.2 The maps: compression, a shared stream, low-rank heads, iterated refinement

A trained network is a composition of maps between these spaces, each map chosen by gradient descent from a parameterized family. Four architectural facts about those maps generate nearly everything this chapter later measures.

The embedding is already a compression. The embedding matrix W_E maps a discrete vocabulary of ∼50,257 symbols into $\mathbb{R}^{768}$. Fifty thousand symbols sharing 768 dimensions is a massive commitment before the first attention layer runs: token geometry is compressed from the very first step, and the unembedding W_U must read the final state back out through the same mismatch.

The residual stream is a shared additive channel. Every attention head and every MLP block *reads* a linear projection of the residual stream, computes, and *adds* its result back; nothing is overwritten, contributions only sum (Elhage et al., 2021). Two consequences follow. First, the final logits decompose *exactly* into a sum of per-component terms - the embedding plus each head's write plus each MLP's write, each read out through W_U. This identity is the basis of direct logit attribution and of the lenses in Section 5.6. Second, the stream is a *limited-bandwidth channel* that all components share: features must compete for room in it. This fact returns in Section 5.4 as the *cause* of superposition, which otherwise arrives as a surprise.

Each attention head is a low-rank bottleneck. The stream width is divided among heads: $d_{\text{head}} = d_{\text{model}}/n_{\text{heads}}$, which for GPT-2 small is $768/12 = 64$. Following Elhage et al. (2021), a head factors into two circuits: the **QK circuit** $W_Q^\top W_K$, which determines the attention pattern (who attends to whom), and the **OV circuit** $W_O W_V$, which determines what is written if a position is attended to. Because W_Q, W_K, W_V each map $d_{\text{model}} \to d_{\text{head}}$ and W_O maps $d_{\text{head}} \to d_{\text{model}}$, both circuit matrices - though they are full $d_{\text{model}} \times d_{\text{model}}$ objects - have **rank at most** d_{head}. That is a hard algebraic bound, not an empirical tendency. The interpretive reading (motivated, not proven): a 64-dimensional channel wedged between a 768-dimensional stream and a 50k-token vocabulary cannot carry arbitrary token-to-token maps; each head must *select* a low-rank slice of features to move. The specialization of heads that Chapter 6 dissects into circuits is this selection pressure made visible.

Depth is iterated refinement, and the hierarchy is soft. Layers are the same map-shape composed many times over the shared stream, and labor divides across depth in a recognizable pattern - detokenization in the earliest layers, feature construction in the early-middle, prediction assembly in the later layers, sharpening at the very end (Lad, Gurnee, and Tegmark, 2024; the evidence appears in Section 5.6). But the hierarchy is soft: deleting or swapping *adjacent* layers of a pretrained model retains 72–95% of its accuracy without retraining (Lad et al., 2024). Middle layers perform incremental, partly interchangeable refinement - a fact to remember whenever a claim assigns a concept a single precise home layer.

5.1.3 The points: activations

An **activation** is a vector a specific layer actually outputs on a specific input: run a sentence through the model, read the residual stream after layer ℓ at one token position, and you hold one point in $\mathbb{R}^{d_{\text{model}}}$. A dataset of N sentences yields a *cloud* of points at every layer - one per token, or one per sequence if reduced (last-token or mean-pooled; the choice changes the object, and recording it saves confusion later: per-token clouds mix punctuation, function words, code fragments, and rare subwords, and are far more heterogeneous than per-sequence clouds).

Two distinctions are now available. A vector is any element of the space; an activation is a vector the network *actually produced*. And activations are *data*, not parameters: the same trained map yields a different cloud on Wikipedia than on Python than on French. Every geometric quantity measured in this chapter is therefore a property of the *pair* (model, input distribution) - a dependence that Section 5.5's closing subsections elevate from caveat to subject matter.

5.2 What Is a Representation?

5.2.1 The definition

With spaces, maps, and activations in hand, the chapter's central term: a **representation** is information encoded in activations that the network's own downstream computation *reads*. Two conditions follow. First, a representation is *used*: some later component's output depends on it. This separates a representation from an incidental correlate that a probe can decode but the network never consults - the correlational/causal distinction that does most of the epistemic work in this field, built into the definition. Second, claims about a representation name a *readout* - a linear probe, the unembedding, a sparse-autoencoder decoder - at a stated location (which layer, which sublayer, which token position), so the claims can be checked.

The content can be carried as a **direction** (sentiment, Section 5.6), a **subspace** (the syntactic subspace of Section 5.5), a **region**, or a **manifold-valued code** (the circular day-of-week feature of Section 5.5). One activation vector simultaneously carries many representations, superposed (Section 5.4): the activation is the *carrier*; the representation is the *content*.

5.2.2 A taxonomy

Several kinds of representations exist. A few mentioned in the literature, along with their location, an example, and how they are read out, are listed below.

Kind	**Where it lives**	**Example**	**Readout**
Token embedding	row of W_E (layer 0)	the vector for *" Paris"*	nearest tokens
Positional code	added at the input	learned/rotary positions	direct inspection
Residual state	stream after layer ℓ	the 768-vector at a position	tuned lens (§5.6)
Attention pattern	per head, per layer	who-attends-to-whom	direct inspection
Feature direction	subspace of the stream	"French-ness"	probe / steering (§5.6)
SAE latent	overcomplete dictionary	"Golden Gate Bridge"	SAE decoder (Ch. 6)
Circuit	across heads and layers	IOI name-movers	patching (Ch. 6)

The rows of the table vary along two axes that decide which tool applies. *Local vs. distributed*: is the content carried by one unit or by a direction spread across units? Superposition

(Section 5.4) is the reason the useful unit of analysis is almost always distributed. *Linear vs. nonlinear*: is the readout an inner product, or does the content live on a curved locus? The productive default is to assume linear first - linear tools are cheap and usually work - and to treat every failure of a linear tool as a finding about geometry, not a null result (Section 5.6 returns to this).

Features. A **feature** is a property of the *input* that the model tracks: present in some inputs, absent in others, with a degree when present. "Contains French," "is a URL," "refers to the Golden Gate Bridge." The feature is the input-side object; its *representation* is how the model encodes it, typically as a direction (Section 5.6). In the toy models of Section 5.4, features are literally input coordinates; in dictionary methods, a feature is whatever a sparse decomposition recovers as one latent (Section 5.5, Chapter 6). For real models there is no direct definition, and the field works with the indirect one - a property a large enough model would dedicate a neuron to - knowing it is circular.

Latent space. The **vector space** $\mathbb{R}^{768}$ is the container of Section 5.1, identical before and after training. The **latent space** is the same container *as organized by the trained model*: some directions move logits, some regions correspond to languages, some subspaces carry syntax. "A direction in latent space" therefore means a direction in $\mathbb{R}^{768}$ together with the claim that the model treats it as meaningful - a claim to be tested, not assumed.

Neuroscience parallel. The question "what is a representation?" is neuroscience's *neural coding* question, inherited nearly verbatim. Rate codes (one neuron's firing rate carries the variable), place codes (which neuron fires carries it), and population codes (a pattern across neurons carries it) map directly onto the local-vs-distributed axis above; and neuroscience, too, learned to require clause (ii) - a decodable signal is not a code until the animal's behavior can be shown to depend on it (the "reading the code" criterion). The parallel is developed throughout the chapter as the tools recur.

5.3 Manifolds and Intrinsic Dimension

5.3.1 Subspace versus manifold; ambient versus intrinsic

A subspace is flat, global, and closed under linear combination. A **manifold** is a possibly curved surface that only *locally* looks like $\mathbb{R}^k$ - a sphere, a spiral, a crumpled sheet. A layer with $d = 768$ units can in principle output any vector in $\mathbb{R}^{768}$, but the activations it produces on real text do not fill that space: they cluster near a curved surface describable locally with far fewer coordinates. (This is a claim about the activation vectors themselves, not about the model's output distribution over the vocabulary, which lives on a separate simplex.) The **intrinsic dimension** is the number of degrees of freedom needed to describe that surface locally: the dimension of the surface, not of the room it sits in.

A practical estimator, Two Nearest Neighbors (TwoNN; Facco et al., 2017), uses only each point's two nearest neighbors to estimate the intrinsic dimension. For point i, let $r_1(i)$ and $r_2(i)$ be the distances to its first and second nearest neighbors, and $\mu_i = r_2(i)/r_1(i)$. This number shrinks as the number of dimensions grows. On a uniform d-dimensional manifold, μ follows a

power-law distribution whose shape depends on the dimension alone. Averaging over points,

$$\hat{d} \;=\; \frac{N}{\sum_{i=1}^{N} \log \mu_i}$$

recovers the dimension from ratios. It does so without assuming the manifold is flat, which is why it is preferred over PCA eigenvalue counting, which asks how many *flat* directions carry variance and systematically overestimates the dimension of a curved cloud. Activation clouds are curved, as the circular features of Section 5.5 show.

5.3.2 The hunchback profile: intrinsic dimension across layers

Ansuini, Laio, Macke, and Zoccolan (2019) measured intrinsic dimension layer-by-layer across trained image networks (AlexNet through ResNet-152) and found a consistent shape they called the *hunchback*: intrinsic dimension *rises* through the early layers, peaks somewhere in the first half of the network, then *falls steadily* toward the output. The ambient dimensions of the layers studied were in the range 10^4–10^5, while the measured intrinsic dimensions were roughly 10 to 100 - two to three orders of magnitude smaller. Furthermore, the intrinsic dimension of the *last hidden layer* predicted generalization: architectures with lower final-layer intrinsic dimension achieved lower test error. Networks trained on randomized labels, which can only memorize, not generalize, showed the opposite signature, with final-layer intrinsic dimension rising rather than compressing (Ansuini et al., 2019).

The interpretation: early layers *unfold* input structure (increasing intrinsic dimension as they disentangle raw correlations), and later layers *discard* decision-irrelevant variation (compressing toward task-relevant subspaces). Representation learning is expansion followed by compression, and you can watch it happen with a nearest-neighbor estimator.

5.3.3 Cautions: sample size, outlier dimensions, and the union-of-manifolds trap

Three practical cautions when measuring intrinsic dimension yourself. First, estimators need enough samples: as a rule of thumb from the TwoNN literature, N points cannot resolve dimensions much above $2 \log_{10} N$. Second, tokenization artifacts and outlier activations (which transformers famously have; see the rogue dimensions of Section 5.5) can dominate nearest-neighbor statistics; a quick look at the raw neighbor distances before trusting the estimate catches most of these cases.

The third caution is the one most often ignored. TwoNN assumes the points come from a *single* manifold of locally uniform density, but the activation cloud is almost certainly a **union of manifolds** of different dimension - token types, syntactic roles, languages, code versus prose (Section 5.5) each plausibly contribute their own component. On such a mixture the pooled estimator returns a blended exponent that need not equal the dimension of *any* component, biased near the seams where a point's neighbors come from the wrong piece. The remedies: estimate per subset that is plausibly one manifold (one token type, one language, one register); check that the number is stable when the sample is halved and the neighborhood scale varied; or use an estimator built for mixtures - Hidalgo (Allegra, Facco, et al., 2020) assigns each point its own local dimension and component. Where subsets disagree with the pooled estimate, the disagreement is the finding.

Neuroscience parallel. The meaningful unit of analysis is the neuron population manifold, not the single neuron. Neuroscience established this through population recordings of motor cortex during reaching movements (Churchland et al., 2012): although individual neurons respond idiosyncratically, the population of hundreds of recorded neurons traces trajectories confined to fewer than ∼20 effective dimensions, and those trajectories, not single-cell tuning curves, predict behavior. In artificial networks the corresponding unit is the *residual stream at a given layer*: the full d_{model}-dimensional activation vector that attention heads and MLP blocks read from and write to. Per-neuron stories can be true but uninformative; the manifold traced by the whole vector is the object that carries the computation. Barrett, Morcos, and Macke (2019) survey this methodological bridge in both directions.

5.4 Superposition: More Features Than Dimensions

5.4.1 The counting problem

Section 5.1 established that the residual stream is a shared, limited-bandwidth channel that every component reads and writes. Let's estimate a count of what must fit through it. A language model tracks hundreds of thousands of distinguishable features of its input - entities, syntactic roles, sentiment, languages, code contexts, factual relations. Sparse dictionaries recover hundreds of thousands to tens of millions of features from real models (Section 5.5). GPT-2 small has only 768 residual dimensions. If each feature required its own orthogonal direction, the model could represent at most 768. It represents far more. Something has to give, and what gives is orthogonality. (Part I previewed this phenomenon from the training side, in its §4.4, as the optimizer's compression at work.)

The **Johnson–Lindenstrauss** phenomenon supplies a loophole: while $\mathbb{R}^d$ contains only d exactly-orthogonal directions, it contains $\exp(\Theta(\epsilon^2 d))$ directions that are pairwise *almost* orthogonal (cosine similarity $\leq \epsilon$). A 768-dimensional space has room for vastly more than 768 nearly-orthogonal feature directions - if the model can tolerate small interference between them.

5.4.2 Three numbers, not two

Sections 5.3 and 5.4 can appear to contradict each other. One says the activation cloud is a low-dimensional manifold, tens of dimensions inside hundreds; the other says the model is starved for dimensions and must pack. If the activation cloud uses thirty dimensions out of 768, where is the shortage?

The resolution is that three different numbers are in play Let s be the number of features *active at once* on a typical input. Let d be the width. Let n be the number of features the model *stores* across its whole input distribution.

Superposition is a statement about n versus d: the dictionary is far larger than the dimension count, so stored directions must overlap.

Intrinsic dimension is, to first approximation, a measurement of s: at any one input, the activation varies only along the directions of the features currently active, so the local dimension of the cloud is the size of the active set, however enormous the dictionary.

A cloud built from sparse combinations is a union of low-dimensional pieces, one per active set, and a local estimator like TwoNN sees the dimension of the piece it stands on.

The picture that stays useful is the ordering

$$s \ll d \ll n.$$

There are roughly tens of active features (s), thousands of dimensions (d), and millions of stored features (n). The gap between s and d keeps interference tolerable: few active features means few collisions. The gap between d and n is what superposition bridges: nearly orthogonal directions are exponentially plentiful, so a large dictionary fits in a small space as long as little of it is active at once. The numbers already reported follow the ordering - intrinsic dimensions of 10 to 100 (Section 5.3), widths of 768 to ∼16,000, dictionaries of up to 34 million features (Section 5.5, Chapter 6). They yield a testable prediction: a layer's intrinsic dimension should agree with the per-token active count at which a sparse autoencoder reconstructs it well ($k \approx 32$–256 in current practice). Two unrelated methods agreeing in order of magnitude is either a coincidence or the ordering at work.

5.4.3 The toy model and its phase transition

Let's define sparsity and importance for a feature. A feature is *sparse* if it is present in only a small fraction of inputs. The sparsity S names the probability that the feature is absent. So density $1-S$ is the fraction of inputs where it appears. Let each feature also carry an *importance*: how much the loss suffers when that feature is reconstructed badly. Both these quantities are properties of the data, not the model.

Sparsity matters for packing because interference is a coincidence cost. Two features that share overlapping directions disturb each other only when both are active in the same input, which happens in roughly $(1-S)^2$ of inputs. As sparsity rises, that collision rate falls quadratically while the value of representing each feature does not fall at all. So there must be a crossover: for dense features, interference is constant and only dedicated dimensions are safe; for sparse features, packing is nearly free.

Before the crossover: features prefer approximately orthogonal, dedicated directions. After the crossover: features can share directions, because collisions are sufficiently rare.

Here's an experiment confirming such a crossover. It takes n features with geometrically decaying importance, compresses them through a $d \ll n$ bottleneck with a linear map, recovers them with the same map transposed plus a bias and a ReLU, trains to minimize importance-weighted reconstruction error, and sweeps the sparsity. At $S \approx 0$ the model behaves like principal component analysis: it keeps the d most important features, one per dimension, and drops the rest. As sparsity rises it switches to packing more features than dimensions into overlapping directions, and the packed geometries are strikingly regular: antipodal pairs first, then triangles, pentagons, tetrahedra. Remove the ReLU and the transition never happens: a linear readout cannot filter out small interference, so the linear model keeps the top d features at every sparsity. This is the toy-model result of *Toy Models of Superposition* (Elhage et al., 2022).

Toy is meant in the physicist's sense - the smallest system that exhibits the phenomenon, with everything else stripped away: no attention, no depth, no text; features are literally input coordinates, so the mechanism (benefit versus collision cost) is visible without confounds. Superposition is not a defect. It is the rational allocation of scarce channel bandwidth, and it emerges from gradient descent in a model small enough to write in five lines.

5.4.4 Two consequences that shape the field

Polysemantic neurons are expected. Section 5.1 noted that the neuron basis is just one basis, with no mathematical privilege. If features are directions in superposition, then any individual neuron (basis direction) will typically have nonzero overlap with many features. A neuron firing for "academic citations, Korean text, and HTTP requests" is not broken; it is the shadow of several features, none of which is axis-aligned. Sixty years after neuroscience's "grandmother cell" debate, artificial networks reproduced the same answer: mostly, there is no grandmother neuron.

The unit of analysis must be the feature, not the neuron. If the concepts are packed into non-axis-aligned, overcomplete directions, then recovering them is a *sparse dictionary learning* problem: find the overcomplete basis in which activations become sparse, interpretable combinations. That is precisely what sparse autoencoders do. Section 5.5 reports what they find about the shape of the space; Chapter 6 develops the method itself.

Neuroscience parallel. Superposition has a biological counterpart: **mixed selectivity**. Rigotti et al. (2013) showed that individual prefrontal cortex neurons respond to nonlinear conjunctions of task variables (stimulus × rule × response) rather than to any single variable, and that this mixing is what a high-capacity distributed code looks like when viewed one neuron at a time. Polysemanticity in artificial networks is the same observation: the single unit looks noisy because the code is carried by directions that cut across units. The toolkit now runs in both directions - sparse-decomposition methods developed for transformer activations are being applied to real neural recordings to unmix biological population codes into interpretable components.

5.5 The Measured Shape of the Space

The previous two sections established what kind of object the activation cloud is - a low-dimensional, heterogeneous union of manifolds carrying superposed features. This section reports what the space actually looks like when measured at scale, and the caveats that keep those measurements honest.

5.5.1 What sparse dictionaries reveal

Sparse autoencoders trained on activations recover overcomplete dictionaries of directions far more monosemantic than neurons (Bricken et al., 2023; Cunningham et al., 2024) - direct empirical confirmation of Section 5.4's prediction that the features outnumber the dimensions and are not axis-aligned. At production scale, dictionaries of up to ∼34 million features have been extracted from the middle-layer residual stream of a frontier model (Claude 3 Sonnet), including the now-famous Golden Gate Bridge feature, features for code bugs, and features that fire across languages and modalities (Templeton et al., 2024). Three structural findings matter for geometry:

- **Feature splitting.** Enlarge the dictionary and one coarse feature splits into several finer ones - "San Francisco" into neighborhoods, a punctuation feature into cases. The feature set you recover depends on the resolution you request; Chapter 8 lists the resulting *resolution illusion* among the field's standing epistemic hazards.

- **Structure at three scales.** The extracted feature cloud is not amorphous (Li et al., 2025): at the *atomic* scale, parallelogram and trapezoid "crystals" generalize the king−man+woman≈queen relation (visible once distractor dimensions such as word length are removed with linear discriminant analysis); at an intermediate scale, features cluster into spatially modular *lobes* - mathematics and code features sit together; at the largest scale the cloud is anisotropic with an eigenvalue power law steepest in middle layers.
- **Features need not be one-dimensional.** Some representations are irreducibly multi-dimensional: days of the week and months of the year occupy *circles* in activation space, verified causally by intervention (Engels et al., 2025). A loop is not a line - these are manifold-valued representations in exactly the sense of Section 5.2's definition.

Here the field is split, and Chapter 8's evidence standard applies to us as much as to anyone: whether SAE features reflect the *model* or the *dictionary* is contested. A nontrivial "dark matter" of SAE reconstruction error resists decomposition (Engels, Riggs, and Tegmark, 2024); hierarchical concepts get *absorbed* into child features, so a seemingly monosemantic feature silently fails to fire (Chanin et al., 2024); and on many known-concept tasks SAE probes do not consistently beat a logistic-regression baseline (Kantamneni et al., 2025; Karvonen et al., 2025). The defensible position: SAEs are a powerful *discovery* tool for structure you did not know to look for, not yet a verified coordinate system.

5.5.2 Anisotropy: the cloud is a narrow cone

The activation cloud does not surround the origin evenly. Contextual embedding spaces are **anisotropic** in every layer of every model measured, increasingly so with depth: the average cosine similarity between the representations of *randomly chosen* words in GPT-2's last layer is roughly 0.63, where isotropy would predict roughly zero (Ethayarajh, 2019). Part of the cause is training itself - the likelihood objective pushes embeddings into a narrow cone (the *representation degeneration* problem; Gao et al., 2019) - and part is concentrated in a handful of **rogue dimensions**: one to three coordinates of such large magnitude that they dominate cosine similarity while carrying little of the behaviorally relevant signal (Timkey and van Schijndel, 2021). Standardizing those dimensions away markedly improves agreement between representational similarity and human similarity judgments.

A habit that saves grief: before leaning on a raw cosine similarity, a raw PCA, or a raw nearest-neighbor query over transformer activations, it helps to check what the top few dimensions are doing. Otherwise the measurement may be reading the cone and the rogue coordinates rather than the semantics.

5.5.3 What two-dimensional pictures cannot show

Much of what the community "knows" about latent spaces arrives through two-dimensional pictures, so the failure modes of the picture-making tools are part of the geometry curriculum. t-SNE (van der Maaten and Hinton, 2008) preserves local neighborhoods at the deliberate expense of global arrangement: cluster *sizes* in a t-SNE plot mean nothing (the algorithm equalizes density), inter-cluster *distances* are often meaningless, results depend strongly on the perplexity setting, and pure noise can be made to look clustered (Wattenberg, Viégas, and Johnson, 2016). UMAP's reputation for better global structure is largely attributable to its default initialization rather than its objective - initialize t-SNE the same way and the difference mostly disappears

(Kobak and Linderman, 2021). PCA is linear, so it cannot unfold a curved manifold at all (Section 5.3). The stance we take in this book: *an embedding scatterplot is a hypothesis-generating device rather than evidence.* Structure it suggests earns belief once it has been re-established quantitatively, by probe, intervention, or explicit geometry.

5.5.4 How much of the shape is universal?

Is the geometry just described a fact about one trained model, or about models in general? The evidence points to *partial, measurable* universality. Independently trained networks of the same architecture can be connected by a *single learned affine map* at little accuracy cost (model stitching; Lenc and Vedaldi, 2015; Bansal, Nakkiran, and Barak, 2021) - operationally, their representations are the same up to an affine change of coordinates. Encoding each point by its similarities to a set of anchor points (*relative representations*) makes independently trained latent spaces comparable with no training at all (Moschella et al., 2023). Pushed further: text-embedding spaces from different encoders can be translated into one another *without any paired data*, adversarially aligning both to a shared latent geometry (vec2vec; Jha et al., 2025). And the *Platonic Representation Hypothesis* (Huh et al., 2024) argues that as models grow more capable, their representational geometries converge - across architectures and even across modalities - toward a shared statistical model of the world, with measured alignment rising with scale.

The bounds matter as much as the claims. Only ∼1–5% of neurons in GPT-2 models trained from different seeds are "universal" in the sense of consistent co-activation (Gurnee et al., 2024): universality lives at the level of directions and features, not units. A high centered-kernel-alignment (CKA) score does not imply functional equivalence, and a single outlier point can move it drastically (Davari et al., 2023). And the Platonic convergence trend weakens under some metric calibrations while surviving under local-neighborhood measures. Summary: latent geometry is convergent enough that cross-model translation works far better than chance, and divergent enough that no "one true representation" claim survives the measurements.

Neuroscience parallel. Representational similarity analysis - comparing systems by the geometry of their response patterns rather than unit-by-unit - was built in systems neuroscience (Kriegeskorte, Mur, and Bandettini, 2008) to compare brains, models, and species on common ground. CKA, SVCCA, and the alignment metrics of this subsection are its machine-learning descendants, and the toolkit now runs in both directions: the same similarity analyses are used to ask whether a language model's geometry matches cortical recordings during language comprehension.

5.6 Reading the Space: Probes and Lenses

Everything in this section is a **projection**: a map from the latent space to something a human can check. A probe projects onto a candidate concept direction; the logit lens projects onto the vocabulary; a stitching layer projects one frame into another; an SAE decoder projects onto an overcomplete dictionary. Keeping the word *projection* in view unifies what otherwise looks like a bag of tricks - and keeps visible that every readout answers only the question it was built to ask.

5.6.1 The linear representation hypothesis

The **linear representation hypothesis** is the claim that many high-level concepts correspond to *directions* in activation space: the model represents "French-ness" or "truthfulness" or "past tense" as a vector, and the degree to which a concept is active is (approximately) an inner product. Three families of evidence, in increasing order of strength:

- **Probing (correlational).** A linear classifier on frozen activations decodes the concept far above chance - often near-perfectly for syntactic and factual properties by the middle layers.
- **Arithmetic (suggestive).** The word-embedding classic (Mikolov et al., 2013): king $-$ man $+$ woman $\approx$ queen. Concept offsets compose additively more often than chance would allow, though the standard evaluation excludes the query words from the answer set, and without that exclusion the nearest neighbor is usually one of the inputs (Linzen, 2016).
- **Steering (causal).** Adding a concept direction to the residual stream at inference time *changes model behavior* in the predicted way. This is the evidence that matters, and Chapter 7 is devoted to it.

The correlational/causal distinction matters most right here: a probe can succeed by reading a direction the model never uses downstream. Probing tells you information is *present*; only intervention tells you it is *used*. In the vocabulary of Section 5.2, probing establishes that the content is *decodable*; only intervention establishes that it is *used*. This distinction guards against the most common inferential error in the field.

Why should linearity hold at all? Part I's training-dynamics chapter supplies a candidate answer: theoretical work shows that the implicit bias of gradient descent, combined with the log-odds structure of the next-token objective, provably yields linear concept encodings in simplified settings (Jiang et al., 2024). If that account is right, the hypothesis is a fingerprint of the optimizer, not an architectural accident - and it could in principle be strengthened or weakened by training choices (Part I, §4.4).

5.6.2 What "direction" means depends on the inner product

Park, Choe, and Veitch (2023) pointed out a subtlety that resolves a lot of confusion. "Concept = direction" is ambiguous until you fix an inner product, and the natural choices disagree. There is an *unembedding* (measurement) view, the direction whose dot product with the final representation moves the output logits for concept-related tokens, and an *embedding* (intervention) view, the direction you add to activations to change the concept. In the raw Euclidean geometry of activation space these two directions are generally *not* the same vector.

Their resolution: define a **causal inner product** under which counterfactually independent concepts become orthogonal. Concretely, they show a transformation (estimable from the unembedding matrix's covariance) that aligns the measurement and intervention geometries. Under the right metric, probing directions and steering directions unify - and the fact that a specific metric is needed is itself evidence that the model's "natural" geometry is not the Euclidean one we plot.

5.6.3 Where linearity fails

The linear representation hypothesis is a strong default, not a law. Documented exceptions: **circular features**, where days of the week and months of the year sit on closed loops (verified causally by intervention experiments; Engels et al., 2025, and Section 5.5), and a loop is not a line; **magnitude coding**, where numeric quantity appears to use nonlinear encodings in some models; and **conditional features**, directions whose meaning changes with context, which behave linearly only within a regime. The productive stance: assume linearity first because linear tools are cheap and often work; treat every failure of a linear tool as a finding about geometry, not just a null result.

5.6.4 The model's own projection: logit lens and tuned lens

The unembedding W_U is the one projection the *model itself* uses: it maps the final residual state to vocabulary logits. The **logit lens** (nostalgebraist, 2020) simply applies that same projection to *intermediate* states: read the residual stream after layer ℓ, apply the final layer norm and W_U, and obtain a per-layer trajectory of "what the model would predict if it stopped here." On GPT-2-class models the trajectory is revealing: the eventual top prediction often appears and stabilizes several layers before the output. But the tool has known failure modes - early layers decode to noise, the readout is systematically biased (intermediate states are not in the basis W_U expects), and the trick transfers poorly to some model families.

The **tuned lens** (Belrose et al., 2023) repairs this by learning, for each layer, a small affine *translator* that maps layer-ℓ states into the frame the unembedding expects before projecting. The result is a far more reliable readout, and an unbiased one in the estimator's sense: each layer's translator is trained to match the model's final distribution, so its errors are not systematically tilted the way the raw unembedding's early-layer readouts are. It works on models up to ∼20B parameters, and its attributions demonstrably track the features the model itself uses rather than artifacts of the probe.

The two lenses together correct a common oversimplification. Under the logit lens, predictions seem to "form late." The tuned-lens picture is subtler and now better supported: much predictive information is present *early* but superposed and misaligned with the unembedding direction - invisible to the raw projection, recoverable by the calibrated one (Belrose et al., 2023; Lad et al., 2024). Depth-wise, the stages of Section 5.1 - detokenization, feature construction, prediction ensembling, residual sharpening - describe when each kind of content becomes *readable*, which is not the same as when it first exists. The lesson generalizes: *where a representation appears depends on the projection you read it with.* That is Section 5.4's superposition speaking, and it is why this book states readouts explicitly.

5.6.5 Mapping between frames: stitching

The tuned-lens translator is a special case of something general. A **stitching layer** is a learned affine map that carries one frame into another: layer i's coordinates into layer j's, or model A's frame into model B's (Lenc and Vedaldi, 2015; Bansal, Nakkiran, and Barak, 2021). That a single affine map usually suffices - across seeds, widths, and sometimes architectures (Section 5.5) - is the operational meaning of "same representation, different coordinates," and it is what licenses moving directions and features between layers and models instead of rediscovering them; SAE features now transfer between language models the same way.

Neuroscience parallel. The question "what does this unit represent?" comes from receptive-field mapping. Hubel and Wiesel (1959; 1962) characterized neurons in primary visual cortex by the stimulus that drove them (oriented edges at particular retinal locations), and visual neuroscience spent the following decades refining that question for higher areas. Mechanistic interpretability asked the identical question of artificial units (feature visualization, maximally-activating dataset examples) and hit the identical wall: units in higher layers respond to bewildering mixtures of stimuli. Both fields resolved it the same way: stop analyzing single units, analyze directions in the population activity space - and read them out with learned linear decoders, which is what the tuned lens is. In a transformer, the analogue of a receptive field is a *direction in the residual stream* together with the set of inputs that push activations along it. Lindsay (2021) documents the full round trip between the two fields.

The measurements so far characterize the space on its training distribution, but every geometric quantity in this chapter is a property of the pair (model, input distribution) - Section 5.1 ended on that warning. The next three subsections vary the input on purpose: the same trained network reorganizes its latent space depending on what flows through it, and the reorganization has reproducible structure.

5.6.6 Languages: shared middle, specific edges

Multilingual models converge on a consistent depth-wise arrangement: *language-specific at the edges, language-shared in the middle.* Reading intermediate layers of Llama-2 with the logit lens on translation tasks, middle layers decode the semantically correct next token but in an English-leaning form before late layers rotate it into the input language - evidence of a shared concept space, biased toward the dominant pretraining language (Wendler et al., 2024). The pattern generalizes beyond language pairs: semantically equivalent inputs across languages *and modalities* (code, arithmetic) land near one another in intermediate layers, and interventions applied in that shared space transfer across languages - the *semantic hub* picture (Wu et al., 2025). Geometrically, languages occupy similar subspaces offset by a **language-mean direction**: subtracting a language's centroid removes language identity while preserving content, and *shifting* a representation by another language's mean induces output in that language (Chang, Tu, and Bergen, 2022; Libovický et al., 2020). Complementarily, *language-specific neurons* concentrate in the top and bottom layers, and activating or suppressing them causally steers the output language (Tang et al., 2024).

Two cautions. The "English" character of the middle layers is a readout artifact of the dominant pretraining language - evidence for a biased shared space, not for the claim that the model "thinks in English." And concept-direction transfer across languages, while real, is often limited to related languages; no single universal language direction has survived testing.

5.6.7 Code versus prose

Code participates in the shared semantic hub - middle layers place a Python snippet near its natural-language description (Wu et al., 2025) - but code also carries structure prose lacks, and the structure is recoverable from the geometry. A low-dimensional **syntactic subspace** in code-model activations suffices to reconstruct the full abstract syntax tree of the input program, with most of the syntactic information concentrated in middle layers (López et al., 2022) - a subspace-valued representation in the sense of Section 5.2. For program *state*, the sharpest result concerns variable binding: models attach reusable **binding-ID vectors** to entities and

their attributes; the IDs form a subspace of their own, transfer across tasks, and pass causal tests, in models from Pythia to Llama (Feng and Steinhardt, 2024). Fine-tuning strengthens this existing mechanism rather than inventing a new one (Prakash et al., 2024). The evidence draws a clear boundary: name-to-value binding is established; a faithful internal simulation of full program execution is not, and current evidence does not support claiming it.

5.6.8 Registers and formats: what is and is not established

At finer grain the evidence thins. What is established: sparse dictionaries surface discrete, causally active features for specific domains and formats - a code-bug feature whose activation makes the model *write* buggy code, features for base64 blobs and DNA motifs, per-script language features whose clamping switches the output language (Templeton et al., 2024). So domain and format are represented, at least in part, as recoverable feature-level structure. What is *not* established: that prose and poetry, or formal and casual register, occupy cleanly separable subspaces - plausible, uninvestigated at the standard this book requires. And for structured input/output - JSON, tables, schema adherence - there is as of this writing *no mechanistic account at all*: the literature is engineering (constrained decoding, format benchmarks), and the only representational handle is indirect (the format features above, and the observation that JSON key-value structure is an instance of the entity-binding problem). These are open problems, listed as such below, not results.

5.7 Applying Pressure to the Latent Space

So far the space has been described and read. This section collects what can be *done* to it: the moves that measurably change how a frozen latent space is used, and the levers that shape what forms in it during training. The division of labor with the rest of the book: Chapter 7 teaches the engineering (doses, evaluations, guarantees), and Part I's Section 4.4 the training-loop details; this section is the catalog, with the evidence status of each item.

5.7.1 Finding a direction: contrast pairs

The workhorse trick is the **contrast pair**. Write two small sets of prompts that differ only in the target concept (honest versus dishonest answers, French versus English, harmful versus harmless instructions), run both through the frozen model, and take the difference of mean residual vectors at one layer: $\hat{v} = \bar{h}_\ell^+ - \bar{h}_\ell^-$. No training, no labels beyond the pairing, and the result is usually a usable concept direction - the method behind activation addition and representation engineering, developed in Chapter 7 (Turner et al., 2023; Zou et al., 2023).

Three calibrations decide whether the direction is any good, and each is a result of this chapter applied. *Which layer*: middle layers usually work best - early layers are too input-bound, late layers too output-committed, matching the intrinsic-dimension profile of Section 5.3. *Which metric*: the direction that probes best and the direction that steers best need not coincide, because "direction" is inner-product-dependent (Section 5.6). And *probe success is not steering success*: presence is not use - the program `concept_direction_probe_and_steer.py` is built around exactly this divergence.

5.7.2 Five moves that work on a frozen space

Each of these has at least one peer-reviewed or widely replicated result behind it; each also has a named failure mode.

- **Add a direction.** $h_\ell \leftarrow h_\ell + \alpha\hat{v}$ at inference time shifts behavior along the concept (Turner et al., 2023). Failure mode: dose - too large an α produces incoherence, not a stronger effect (Chapter 7).
- **Ablate a direction.** Project a concept out of every residual write, $h \leftarrow (I - \hat{v}\hat{v}^\top)h$. Applied to the refusal direction, this largely disables refusal across 13 model families while general capabilities stay intact (Arditi et al., 2024) - the cleanest single-direction result in the literature, and Chapter 7 discusses what it implies about safety training. Failure mode: only the linearly encoded part is removed; nonlinear or distributed encodings pass through (Section 5.6's exceptions).
- **Clamp a feature.** Hold one sparse-autoencoder feature at a fixed activation while generating. Clamping the Golden Gate Bridge feature made a frontier model discuss the bridge unprompted (Templeton et al., 2024) - control at the level of a named feature, not just a direction. Failure mode: feature splitting means "the" feature is a resolution-dependent choice (Sections 5.5 and 6.1).
- **Shift by a class mean.** Replacing a language's centroid with another language's switches the output language (Section 5.5); the same move applies to registers and formats wherever they are mean-separated. Failure mode: a mean is a crude summary of a region - variance along every other direction is untouched.
- **Translate the frame.** A learned affine map carries a direction found at one layer - or in one model - into another's coordinates (stitching, Section 5.6; cross-model translation, Section 5.5). Failure mode: the map is only as good as the local linearity it assumes, and it degrades across architecture changes.

5.7.3 Pressure during formation: what forms is negotiable

The deeper trick is upstream: apply pressure *while* the representations form, and the geometry this chapter measures comes out different. Part I's Section 4.4 surveys this properly; the short version, from the representation side:

- *Incidental pressure is already operating.* Weight decay, dropout, and width silently decide how much feature-packing and how many backup circuits a trained model contains - legibility is set by choices nobody audited (Part I, §4.4, regime 1).
- *Legibility can be bought.* Codebook bottlenecks make hidden states discrete and enumerable (Tamkin et al., 2023); Backpack-style architectures fix a non-negative sense-vector decomposition into the model itself (Hewitt et al., 2023); training adjustments can push toy models into one-meaning-per-neuron solutions at little loss penalty (Jermyn et al., 2022). The recurring cost is a capability tax, and the recurring trap is optimizing a legibility proxy while the computation relocates elsewhere (the cautionary episode of Part I, §4.4).

- *Formation can be targeted.* Gradient routing confines a designated capability to a designated subregion during training, so it can later be removed cleanly (Cloud et al., 2024). Concept-ablation fine-tuning projects out unwanted concept directions *during* training and largely prevents emergent misalignment (Casademunt et al., 2025; Betley et al., 2025). Persona vectors identify trait directions in advance, letting trainers screen data by its pressure on a trait, or absorb that pressure during fine-tuning so the deployed model never encodes it (Chen et al., 2025); the same directions can now be traced as they form across pretraining checkpoints (Moskvoretskii et al., 2026).

- *Architecture is a formation lever.* The residual stream Section 5.1 is a carrier of representations and its design has an important influence that shapes how representations form.

  The mixing between the residual stream and the outputs of a layer (or sublayers - attention, mlp) - is done by normalization. Where the normalization applies affects how the gradients flow back, and how the representations flow forward.

  Say a layer has input x and an output y=f(x). The normalization can be applied on y alone and the result added to x. This allows gradients to flow back to x without going through the normalization step. Or the normalization can be applied after x and y are added together. This forces the gradient to flow through the normalization step.

  In the first, called Pre-Norm, where normalization occurs inside the residual branch, training is stable but lets deep layers drift toward redundancy, hidden states from the different x collide, causing representation collapse.

  In the second, called Post-Norm, where layer normalization is applied after the residual addition - the representations of different layers stay more distinct , however each layer's heavy dependence on its residual branch amplifies small parameter perturbations into large output disturbances (Liu et al., 2020), which can destabilize training.

  Measuring these effects can be done with cross-layer similarity, anisotropy, and the marginal intrinsic dimension of the deep layers - discussed earlier.

  This presents a tradeoff - a seesaw, and addressing it led to Hyper-Connections: Zhu et al. (2024) replace the fixed identity path (x, y added directly) with several residual streams mixed by learned, input-dependent matrices. Active research continues to address the same tradeoff in 2026, with manifold-constrained HyperConnections (mHC) and Gated Residuals (GR).

Status note: the frozen-space moves are established; the formation levers are young, validated mostly at small scale. What is missing is measurement: nobody has yet priced a formation lever in this chapter's currency - intrinsic-dimension profiles, anisotropy, dictionary quality - against its capability cost. The Research Directions take that up.

5.8 From Representations to Conceptors

One further object, introduced here because Chapter 7 steers with it. A representation encodes *content*: a direction, subspace, or manifold-valued code that downstream computation reads. A **conceptor** (Jaeger, 2014) instead summarizes *occupancy*: it is a soft projection operator

$$C \;=\; R\,(R+\alpha^{-2}I)^{-1}, \qquad R=\tfrac{1}{N}\textstyle\sum_i x_i x_i^{\top},$$

computed from the correlation matrix R of a cloud of activation states. Geometrically, C is an ellipsoid fitted to the cloud - the region of latent space a whole class of states occupies when the network processes a given pattern, with the aperture α setting how tightly - and applying C to a state softly projects it into that region. Because conceptors are operators they compose: AND, OR, and NOT combine regions, which no single direction can do.

The consequence for control, developed in Chapter 7: steering vectors add a direction to the stream; conceptor steering projects the stream into a region. The two are complementary and fail differently. The program `conceptor_steering.py` lets you feel the difference by hand.

Takeaways

- The ladder holds the vocabulary: vector spaces are empty containers; learned maps (compressing embedding, additive residual stream, rank-$\leq d_{\text{head}}$ heads) give them structure; activations are the points actually produced; representations are the *content* those points carry - direction-, subspace-, or manifold-valued, and defined by being *used* downstream, not merely decodable.
- Activations occupy manifolds of intrinsic dimension 10–100 inside spaces of ambient dimension 10^3–10^5; the profile rises then falls across layers, and final-layer compression tracks generalization (Ansuini et al., 2019). The cloud is a *union* of manifolds, so single global dimension estimates are population summaries, not facts about any component.
- Superposition explains polysemanticity as rational packing of sparse features into limited channel bandwidth (Elhage et al., 2022), and it forces the field's unit of analysis from neurons to features.
- Measured at scale, the space has shape: feature crystals, lobes, and circular features (Li et al., 2025; Engels et al., 2025); a strong anisotropic cone with rogue dimensions that corrupt naive cosine similarity (Ethayarajh, 2019; Timkey and van Schijndel, 2021); partial cross-model universality - and the shape reorganizes with the input: language-shared middles with language-specific edges, syntactic subspaces and binding IDs for code.
- Every readout is a projection, and what you see depends on which one you use: probes read candidate directions (present $\neq$ used; "direction" is metric-dependent, Park et al., 2023); the logit lens reads the model's own unembedding and is biased early; the tuned lens calibrates it; affine stitching moves representations between frames.
- The space is not only readable but pressable: contrast-pair directions, addition and ablation, feature clamping, mean shifts, and frame translation all work on a frozen model, and training-time levers (gradient routing, concept ablation during fine-tuning, persona screening) shape what forms in the first place.

- A conceptor summarizes *where* a class of states lives rather than *what* is encoded, the bridge from this chapter's geometry to Chapter 7's control.

Research Directions

From §5.3: Does fine-tuning move the hunchback?

Measure the layer-wise intrinsic-dimension profile of a base model and its instruction-tuned or RLHF'd sibling (e.g., an OLMo or Qwen base/instruct pair) on identical text. Three concrete questions: (1) Does post-training shift intrinsic dimension mostly in late layers, matching the intuition that alignment tuning edits behavior-adjacent representations? (2) Do behavioral changes from fine-tuning localize to the layers where intrinsic dimension shifted most? (3) Can an intrinsic-dimension-profile diff serve as a cheap first-pass "what changed" scan for model diffing - a question the applied interpretability community currently attacks with far more expensive tools?

From §5.4: Superposition in mixture-of-experts models

Mixture-of-experts architectures route tokens to specialized experts, which in principle relieves dimensional pressure: an expert that sees only code tokens needn't share capacity with poetry features. Do experts therefore exhibit *less* superposition (cleaner, more monosemantic neurons) than a dense model of matched capacity? Measure interference statistics and sparse-autoencoder reconstruction quality per-expert on an open mixture-of-experts model (e.g., OLMoE or Mixtral-class). Either answer is informative: less superposition means routing is an implicit interpretability tool; equal superposition means the packing pressure comes from something other than raw dimension scarcity - itself a major clue about why superposition happens in real models.

From §5.5 and §5.8: Do occupancy geometry and dictionaries agree?

Compute conceptors for two registers (code versus prose) from GPT-2 residual states, and measure the overlap of their ellipsoids (via the conceptor AND operation) against the co-activation statistics of SAE features for the same registers. Two instruments, one question - how distinct are the registers? - with no guarantee of the same answer. Agreement would cross-validate both tools; disagreement would locate exactly what dictionary features miss about occupancy geometry, or vice versa.

From §5.6: Do concept directions transfer to the wild?

Train a truth/falsehood direction on curated statement datasets (the standard setup), then test it on naturally occurring model errors, sycophantic agreements, and real transcripts where the model asserts something false. Report transfer AUROC *and* calibration under distribution shift, plus recall at a 1% false-positive budget - the deployment-relevant number. A deception direction that generalizes to unseen situations would be one of the most valuable artifacts interpretability could produce; every failure mode documented along the way is itself publishable evidence about how concept representations fragment.

From §5.6: Does the resolving layer predict editability?

Build a tuned lens for a small open model. For a battery of factual prompts, record the layer at which the correct answer first becomes readable, then attempt the corresponding fact edit or steering intervention at layers around it. Does the resolving layer predict where interventions succeed? A positive answer would turn a cheap readout into a targeting tool for Chapter 7's methods; a negative one would be direct evidence that readability and causal locus dissociate - itself a finding about superposition.

From §5.7: Price a formation lever in geometric currency

Take one formation-pressure lever - gradient routing on a designated capability, persona-vector screening of fine-tuning data, or the residual geometry itself (Pre- versus Post-Norm, or Hyper-Connections) - and one measurement from this chapter: the intrinsic-dimension profile, the anisotropy, or sparse-dictionary reconstruction quality. Train matched small models with and without the lever and report both the capability cost and the geometric change. The interpretability tax of Part I's §4.4 has never been measured geometrically; either answer - the levers move the geometry cheaply, or they do not - says where the field's effort belongs.

Programs for Chapter 5

`intrinsic_dimension_profile.py` Load GPT-2 small via `transformers`. Run ∼2,000 sentences through the model, caching the residual stream at each of the 12 layers (one vector per sequence, e.g. last-token or mean-pooled). Implement TwoNN in ∼15 lines of numpy: pairwise distances, two nearest neighbors, $\hat{d} = N/\sum_i \log \mu_i$. Plot intrinsic dimension vs. layer; you should see a rise-then-fall profile. Then rerun with shuffled-word sentences and compare. *Extension:* plot the layer-6 estimate as a function of sample count N to see estimator saturation firsthand; then split the sentences by type (code vs. prose) and estimate each subset separately to see the multi-manifold caution of §5.3 in the data. *Type: script + matplotlib. Deps: torch, transformers, numpy.*

`concept_direction_probe_and_steer.py` Build a sentiment direction in GPT-2 small: run 50 positive-template and 50 negative-template sentences, take the difference of mean layer-6 residuals. Evaluate it *both* ways. (1) Probe: held-out classification accuracy using only the projection onto the direction. (2) Steer: add $\pm\alpha \cdot \hat{v}$ to layer-6 activations during generation from neutral prompts and score continuation sentiment with a lexicon. Plot probe accuracy and steering effect vs. layer - observing that the best-probing layer is often *not* the best-steering layer is what the exercise is for. *Type: script + matplotlib. Deps: torch, transformers, numpy.*

`superposition_phase_transition.py` Reimplement the core toy model of Elhage et al. (2022) in ∼80 lines of torch: n=20 features, d=5 hidden dimensions, importance decaying geometrically, feature sparsity S swept from 0 to 0.999. Train $W \in \mathbb{R}^{5\times 20}$ with reconstruction loss; for each S, plot (a) the number of features represented (columns of W with norm > 0.5) and (b) the interference matrix $W^\top W$ as a heatmap. You will watch the phase transition and the polygon geometries appear on your own screen - instructive out of all proportion to its code size. *Type: pure torch + matplotlib, no pretrained model needed.*

`entropy_lens.py` Chapter 1's entropy, applied inside the machine (§5.6 made runnable). Read the residual stream after every GPT-2 layer through the model's own unembedding and measure two things per layer: the Shannon entropy of the readout, and $D_{\mathrm{KL}}(\text{final} \,\|\, \text{layer})$ - the divergence of each layer's readout from the model's actual final prediction. The KL curve falls with depth: the prediction forms. The entropy curve does *not* fall monotonically, and that is the lesson: early logit-lens readouts can be confidently wrong (low entropy about the wrong tokens, the lens bias of §5.6), so entropy alone cannot separate confident-right from confident-wrong - pair it with a divergence to a reference readout. *Type: script + matplotlib. Deps: torch, transformers.*

6. Features, Circuits, and Causal Verification

Chapter 5 ended with a problem statement: the concepts a network uses are directions in superposition - overcomplete, non-axis-aligned, invisible to per-neuron inspection. This chapter is about the response.

Structure of this chapter. Section 6.1 covers **sparse autoencoders**, which recover candidate feature directions from superposed activations by dictionary learning. Section 6.2 covers **circuit discovery**: composing recovered parts into documented mechanisms - named algorithms implemented by specific attention heads and layers - through the two best-studied examples, induction heads and the indirect-object circuit. Section 6.3 covers **causal verification**: the experimental methods (ablation, activation patching, causal scrubbing) that test whether a proposed mechanism story describes what the model actually computes. The order is parts → wiring → proof, and the third step is the one that separates this field from storytelling: it is easy to produce an explanation that fits the examples you looked at, and Section 6.3 is about the experiments that can tell you you're wrong. As in every chapter, open problems and programs are collected at the end.

6.1 Sparse Autoencoders: Learning a Dictionary of Features

6.1.1 Dictionary learning on frozen activations

If activations $x \in \mathbb{R}^d$ are sparse combinations of $n \gg d$ feature directions, the recovery problem is classical sparse dictionary learning, and the workhorse solution is a **sparse autoencoder** trained on activations from a *frozen* model:

$$f = \mathrm{ReLU}(W_{\mathrm{enc}}(x - b_{\mathrm{dec}}) + b_{\mathrm{enc}}), \qquad \hat{x} = W_{\mathrm{dec}} f + b_{\mathrm{dec}},$$

with loss $\|x - \hat{x}\|_2^2 + \lambda \|f\|_1$. The decoder columns are the candidate feature directions; the sparse code f says which features are active, and how strongly, on each token. The L_1 penalty (or, in later variants, a hard TopK constraint keeping only the k largest latents) forces each activation to be explained by few features - the inductive bias that matches the superposition hypothesis.

Everything about sparse-autoencoder practice is governed by one trade-off, the **reconstruction–sparsity frontier**: more sparsity means more interpretable features but worse reconstruction, and the behavioral cost of imperfect reconstruction is measured by splicing $\hat{x}$ into the forward pass - running the model with the layer's activation replaced by its reconstruction - and reporting how much the model's loss increases.

6.1.2 What comes out: monosemantic features, at frontier scale

Trained on real language-model activations, sparse autoencoders recover features that are strikingly more interpretable than neurons. Cunningham, Ewart, Riggs, Huben, and Sharkey (2023) showed systematically that sparse-autoencoder features on Pythia-70M activations were more monosemantic than neurons, PCA components, or ICA components under automated interpretability scoring. Anthropic's work on Claude 3 Sonnet (Templeton et al., 2024) scaled the

recipe to a frontier model and recovered features for specific entities (the Golden Gate Bridge feature, famously steerable), code-security patterns, sycophancy, and safety-relevant abstractions - with feature quality improving as dictionaries grew into the millions of latents.

The current scale record illustrates the engineering envelope: Gao et al. (2024, OpenAI) trained a 16-million-latent TopK sparse autoencoder on GPT-4 residual activations over 40 billion tokens. The TopK activation rule eliminates the shrinkage bias of L_1 training (under an L_1 penalty, features systematically underestimate their own magnitudes). They report clean power-law scaling of reconstruction quality with dictionary size and compute, and introduce the evaluation suite - downstream-loss recovery, probe-based feature quality, ablation sparsity - that has become standard. Typical operating points keep $k \approx 32$–256 features active per token out of 10^5–10^7 candidates.

6.1.3 The evaluation problem and known failure modes

There is no ground-truth feature list for a real model, so sparse-autoencoder evaluation leans on proxies: language-model-generated feature explanations scored by their ability to predict activations on held-out text; loss recovered when reconstructions replace activations; human ratings on feature dashboards. Three failure modes recur and deserve names:

- **Dead features.** Latents that never fire; wasteful, and their fraction grows without initialization and resampling tricks.
- **Feature splitting.** Grow the dictionary and one coherent feature ("month names") splits into finer variants (each month). Not wrong, but it means "the" feature set is resolution-dependent - a dictionary-size choice, not a fact about the model.
- **The reconstruction gap.** Even large sparse autoencoders leave residual error ("dark matter"), and the error is not random - it carries structure the dictionary failed to capture. Whatever lives there is invisible to every downstream analysis built on the dictionary.

Neuroscience parallel. Sparse dictionary learning entered neuroscience thirty years before it entered interpretability. Olshausen and Field (1996) showed that imposing sparsity on a linear code for natural images yields V1-like oriented edge filters - sparse coding as a normative theory of biological representation. A sparse autoencoder is the same mathematics pointed at transformer activations instead of images. The loop has now closed: deconvolutional sparse-coding methods are now applied to real neural recordings, decomposing single-neuron activity into interpretable components. When a method transfers in both directions its failure modes transfer too: the dictionary-size dependence above corresponds directly to the basis-count choice in analyzing neural data.

6.2 Circuit Discovery

6.2.1 From features to mechanisms

A **circuit** is a causal subgraph of model components - attention heads, MLP blocks, feature directions - that implements an identifiable behavior. Where a sparse autoencoder names the variables, a circuit is the program fragment: which components read which information from

where, transform it, and write the result for later components to consume. Two documented circuits anchor the field, one simple and universal, one messy and instructive.

6.2.2 Induction heads: the canonical circuit

The behavior: given text containing a repeated pattern $[A][B] \dots [A]$, predict $[B]$. The mechanism, documented by Olsson et al. (2022), is a two-head composition spanning two layers:

1. A **previous-token head** in an early layer copies each token's identity into the following position's residual stream.
2. An **induction head** in a later layer, at the current $[A]$, queries for positions whose *previous-token annotation* matches $[A]$ - finding the earlier occurrence - and copies the token that followed it ($[B]$) into the output.

This is an algorithm - match-and-copy - implemented in attention weights, and it generalizes to patterns never seen in training.

Induction heads also form abruptly, and the timing is one of the most suggestive facts in the field. Early in training (for the models in the study, in a window around 2.5–5 billion tokens) there is a visible bump in the loss curve during which induction heads appear in every model of the family, and *simultaneously* the model's in-context learning ability - measured as loss at late token positions minus loss at early positions - improves sharply (Olsson et al., 2022). In small attention-only models, ablating induction heads removes the majority of in-context learning. The evidence pattern here became a template for the field: correlation in timing across model sizes, then a causal test by ablation.

6.2.3 Indirect object identification: the full picture

The task: *"When Mary and John went to the store, John gave a drink to"* $\rightarrow$ *"Mary"*. Wang, Variengien, Conmy, Shlegeris, and Steinhardt (2022) reverse-engineered how GPT-2 small does this and found a circuit of 26 attention heads in 7 functional classes: duplicate-token heads flag that "John" repeats; S-inhibition heads suppress attention to the repeated name; name-mover heads copy the remaining name to the output. So far, so clean. But the details are a warning label for the whole enterprise. There are **backup name-mover heads** that do little normally but *activate when the primary heads are ablated* - knock out the main pathway and the circuit partially self-repairs. And some heads ("negative name movers") actively push *against* the correct answer.

Wang et al. (2022) also gave the field its quality criteria: **faithfulness** (the circuit alone reproduces the behavior), **completeness** (nothing important was left out, tested by ablating the claimed circuit's complement), and **minimality** (every included component earns its place). Most published circuit analyses, including this one, satisfy them only approximately - stated openly in the paper, and a model for how to report mechanism claims.

Manual circuit analysis of this depth takes expert-months for a three-token behavior in a 124M-parameter model. That cost curve motivated automation: ACDC-style automated pruning of computational graphs (Conmy et al., 2023), and attribution patching (gradient-based linear approximation of patching effects, one forward-backward pass instead of one run per component; Syed et al., 2023). Automation trades verification depth for coverage - which is exactly the trade-off Section 6.3 exists to police.

Neuroscience parallel. The experimental design behind the indirect-object analysis - minimal prompt pairs differing in one computational demand, used to isolate the components responsible - is the **localizer contrast** from functional imaging. Kanwisher and colleagues isolated face-selective cortex by contrasting responses to faces vs. objects (Kanwisher et al., 1997); circuit analysis isolates name-moving heads by contrasting *"John gave to Mary"* with name-swapped variants. The backup name-mover phenomenon likewise has a direct biological ancestor: lesion a brain region and function partially returns over time as other regions compensate. Redundancy after damage is a property of trained networks of any substrate, which is why single-ablation evidence is weak evidence in both fields.

6.3 Causal Verification Methods

6.3.1 The ladder of evidence

Every claim in this chapter lives somewhere on a ladder:

Rung	Question answered	Cost
Observation	does the component correlate with behavior?	free
Probing	is the information present here?	minutes
Ablation	is the component necessary?	cheap
Patching	does it carry *this* information for *this* behavior?	moderate
Scrubbing	is the *whole hypothesis* consistent with behavior?	expensive

Field norms have converged: observation and probing generate hypotheses; ablation and patching test components; scrubbing tests stories. Publishing a mechanism claim supported only by the first two rungs is how interpretability illusions happen.

6.3.2 Activation patching, done carefully

The method: run the model on a *clean* prompt and a *corrupted* prompt (e.g., the indirect-object sentence vs. a name-swapped variant); cache activations from one run; re-run the other while splicing in the cached activation at a single component; measure how much of the behavioral difference the splice restores. Sweep components to map *where* the behavior-relevant information flows.

Zhang and Nanda (2023) showed that the conclusions are sensitive to choices that look innocuous:

- **Direction.** Denoising (patch clean into corrupted; tests sufficiency) and noising (patch corrupted into clean; tests necessity) give genuinely different maps - report which you ran.
- **Metric.** Logit difference between the two candidate answers is usually the right choice; raw probability saturates and hides graded effects; KL divergence answers a subtly different question: how much of the model's *entire* output distribution the patch restores, rather than how much of the specific clean-versus-corrupted answer contrast.
- **Corruption.** Swapping a name is a targeted counterfactual; Gaussian-noising an embedding takes the model off-distribution and can produce artifacts. Once an intervention pushes activations outside the distribution the model was trained on, every measurement

downstream of it confounds "the mechanism you meant to test" with "the model reacting to inputs it has never seen" - and no later analysis step can separate the two.

Ablation is the degenerate patch (splice in nothing). Even there, the reference distribution matters - **zero-ablation** is off-distribution for components with large mean output; **mean-ablation** removes information while staying in-distribution; **resample-ablation** (splice activations from a random other prompt) stays closest to the data distribution and is the noisiest. Running all three and checking agreement is cheap insurance, and the `activation_patching_lab.py` program at the end of this chapter makes the disagreement visible.

6.3.3 Causal scrubbing: testing the story itself

Patching tests components one at a time. **Causal scrubbing** (Chan et al., 2022) tests a complete hypothesis: formalize the claim as a computational graph annotated with which intermediate values matter for what, then resample-replace *everything the hypothesis says doesn't matter* and measure how much behavior survives. A hypothesis that says "only the name-mover pathway matters" licenses scrambling everything else; if performance collapses, the hypothesis was wrong - it claimed irrelevance for something relevant. Behavior preserved under maximal licensed scrambling is the strongest evidence format the field currently has. The caveats: it is expensive, it rejects but never confirms (surviving scrubbing $\neq$ true), and hypotheses need translation into its formalism.

Neuroscience parallel. Ablation *is* the lesion study, and it inherits a century of lesion epistemology. The core lesson, learned in neuroscience first: a lesion reveals what the remaining network does without the region, not what the region did: compensation and redundancy (the backup name-mover heads again) systematically corrupt the inference from deficit to function. The methodological repair was also invented in neuroscience: *reversible* inactivation with within-subject controls (cooling probes, muscimol injection), which corresponds exactly to inference-time ablation with paired prompts - temporary, dose-controlled, and comparable against the same subject's intact behavior.

Takeaways

- Sparse autoencoders turn the superposition hypothesis into a working tool: overcomplete sparse dictionaries recover monosemantic features at scales up to 16M latents (Gao et al., 2024) - with dictionary-size dependence and a persistent reconstruction gap as open caveats.
- Circuits are documented mechanisms: induction heads (clean, universal, tied to a training phase change; Olsson et al., 2022) and the indirect-object circuit (26 heads, backup pathways, self-repair; Wang et al., 2022) bracket the realistic range from elegant to messy.
- Verification is a ladder - probe, ablate, patch, scrub - and metric, direction, and reference-distribution choices materially change conclusions (Zhang & Nanda, 2023). The backup-head phenomenon is the standing reason single-ablation evidence is weak.

Research Directions

From §6.1: Evaluate sparse autoencoders by what they let you do

Current metrics reward plausible-looking features. A harder, better test: does the sparse autoencoder help on a downstream task with an objective answer - eliciting a secretly fine-tuned fact from a model, detecting a known-injected behavior, or beating linear probes at low-false-positive-rate monitoring? Pick one task with unambiguous ground truth, compare sparse-autoencoder features against strong cheap baselines (raw-activation probes, logit lens), and report where the expensive method actually pays. Negative results here are unusually valuable: the field needs to know which uses survive contact with baselines.

From §6.2: Do expensive circuit tools beat guess-and-check?

Attribution graphs and automated circuit discovery produce detailed candidate mechanisms, but the cheap baseline - an experienced person reading model outputs, guessing the mechanism, and testing with a handful of targeted ablations - is rarely benchmarked. Take three behaviors with partially known circuits (indirect-object identification, greater-than comparison, docstring completion), give each method a fixed compute/time budget, and score recovered mechanisms against the literature. If baselines win at 10% of the cost, that reallocates the field's effort; if they lose, we finally have the demonstration that the tooling investment pays.

From §6.3: The cheapest experiment that catches a wrong story

Build model organisms: small transformers trained on algorithmic tasks where the true mechanism is known by construction (or verified exhaustively). Then write deliberately flawed-but-plausible circuit hypotheses and measure which verification methods, at which compute budgets, reject them. The deliverable is a receiver-operating curve for verification methods themselves - evidence-per-dollar for patching configurations vs. scrubbing. Nothing would do more to standardize what "verified" means on a mechanism claim.

Programs for Chapter 6

`sparse_autoencoder_features.py` Cache GPT-2 small layer-6 residual activations over ∼5M tokens of OpenWebText sample (a few GB; one laptop-night). Train a 768 → 8192 TopK sparse autoencoder (k=32) in plain torch - the training loop is under 60 lines. Then build the payoff tool: for any latent, print the 20 highest-activating contexts with the trigger token highlighted. Spend an hour browsing; you will find clean features (a close-quote feature, a "list item" feature) and messy ones. *Extension:* auto-label 100 random features with a local language model and score explanation precision on held-out contexts. *Type: training script + CLI browser. Deps: torch, transformers, numpy.*

`induction_head_finder.py` Generate sequences of 50 random tokens repeated twice. Run GPT-2 small, and for every head compute the *prefix-matching score*: attention mass from position t (in the second half) to the position right after the earlier occurrence of token t. A few heads will light up above 0.5 while the rest sit near zero - you have found the induction heads. Then ablate them (zero their output) and measure loss on the second

half of repeated sequences vs. the first: in-context learning visibly degrades. Total: ∼100 lines with `transformers` hooks. *Type: script + matplotlib.*

`activation_patching_lab.py` Reproduce the core indirect-object result in GPT-2 small: 20 clean/corrupted prompt pairs (name-swapped), patch each head's output at the final token position, plot the logit-difference recovery as a 12×12 heatmap. The name-mover heads (around layers 9–10) will stand out. Then rerun the sweep three times with zero-, mean-, and resample-ablation of the top heads and put the three ranked lists side by side: the ranks *change*. Sitting with that instability teaches more methodology than any paper. *Type: script + matplotlib.*

7. Control and Steering

Everything so far has been read-only: measure the geometry, name the features, map the circuits. This chapter is read-write. If a concept is a direction in activation space (Section 5.6), then adding that direction at inference time should *change what the model does* - and it does, often with surprising precision.

Structure of this chapter. Section 7.1 covers the simple additive method (computing a concept direction from contrast pairs and adding it to the residual stream) together with the area's strongest empirical result, the refusal direction. Section 7.2 generalizes from vectors to **operators**: conceptors (soft projections with a Boolean algebra) and affine steering maps with optimality guarantees. Section 7.3 treats the engineering question that determines whether any of this gets used in practice: when does steering beat the two conventional levers, prompting and fine-tuning?

Steering matters for two distinct reasons. Practically, it is a control surface: cheap, reversible, no retraining. Scientifically, it is the strongest evidence format for representation claims - a direction that *causally moves behavior* when added is a direction the model uses, not just one a probe can read. Every steering success in this chapter doubles as a confirmation of the linear representation hypothesis of Chapter 5.

7.1 Steering by Activation Addition

7.1.1 The additive recipe

The simplest steering method, **activation addition** (Turner et al., 2023), needs no training at all:

1. Choose a **contrast pair**: prompts embodying the target concept and its opposite ("Love" vs. "Hate"; 50 harmful vs. 50 harmless instructions).
2. Compute the difference of residual-stream activations at a chosen layer ℓ - one pair, or a difference of means over many pairs: $\hat{v} = \bar{h}_\ell^+ - \bar{h}_\ell^-$.
3. At inference, add it: $h_\ell \leftarrow h_\ell + \alpha\hat{v}$, for every generated token.

Two hyperparameters do all the work. **Layer**: middle layers usually steer best; early layers are too input-bound, late layers too output-committed (compare the intrinsic-dimension profile of Section 5.3). **Coefficient** α: too small does nothing, too large produces incoherence - the model rambles about the concept instead of shifting style. Serious evaluations report a dose–response curve of behavior change *and* capability retention vs. α, never a single cherry-picked operating point.

Zou et al. (2023) framed this as a general paradigm they called *representation engineering*, with two primitives: **reading vectors** (project activations onto a concept direction to *detect* honesty, emotion, or harmfulness while the model runs) and **control vectors** (add the direction

to *modify*). Detection and control from the same object is the practical payoff of the causal-inner-product story of Section 5.6 - and the detect-then-control loop is the seed of conditional steering, which returns in Section 7.3.

7.1.2 The refusal direction: the cleanest result in the area

Arditi et al. (2024) asked whether refusal (the trained behavior where a chat model declines harmful requests) is linearly represented. The answer is the strongest single-direction result published to date. Across 13 open chat models spanning 1.8B to 72B parameters (Qwen, Llama, and Gemma families), a single direction per model, computed as a difference of means between harmful and harmless instruction activations, mediates refusal in both directions. *Ablating* the direction (projecting it out of every residual write, or orthogonalizing the weights against it once) largely removes refusal of harmful requests while general capabilities remain statistically indistinguishable from baseline (MMLU and ARC essentially unchanged). *Adding* the direction on harmless prompts makes the model refuse to tell you a cake recipe. One direction, found with a few hundred prompts and no gradient steps, functioning as a behavioral switch across every model family tested (Arditi et al., 2024).

Three readings of this result, all of which matter. As *science*: strong causal confirmation that a high-level, safety-relevant behavior is one-dimensional in every current open model - the linear representation hypothesis at its best. As *security*: weight orthogonalization is a training-free jailbreak, which tells you what single-direction safety training is worth against an attacker with weight access. As *methodology*: the paper's evaluation pattern (behavior shift *and* capability retention *and* cross-model replication) is the reporting standard steering work should meet.

7.1.3 Measuring side effects

A steering intervention that "works" can fail three quieter ways: **capability tax** (perplexity or benchmark degradation while steered), **off-target drift** (an honesty vector that also shifts formality or verbosity - expected whenever concept directions are correlated rather than orthogonal, which superposition guarantees), and **incoherence masquerading as success** (a "refusal-suppression" vector that works by making the model too confused to refuse). The minimum credible evaluation: dose–response on the target behavior, a fixed capability suite at the operating α, and a read of off-target behavior deltas.

Neuroscience parallel. Steering by adding to a population code was demonstrated in the brain before it was demonstrated in transformers. Salzman, Britten, and Newsome (1990) injected microstimulation current into direction-selective neuron clusters in visual area MT while monkeys judged motion direction - and the animals' perceptual reports shifted toward the stimulated direction. Driving a population along its coding axis changes downstream behavior: that is activation addition, three decades early. The transformer analogue of the stimulating electrode is the hook that adds $\alpha\hat{v}$ to the residual stream; the analogue of reversible chemical inactivation (muscimol, cooling) is inference-time directional ablation: temporary, dose-controlled, and removable.

7.2 Steering with Conceptors and Affine Maps

7.2.1 From vectors to operators

Additive steering applies the same nudge to every activation regardless of where it already is. Before generalizing, name the pieces. A **projection** onto a unit direction $\hat{v}$ is the rank-one matrix $\hat{v}\hat{v}^\top$: applied to h it returns $(\hat{v}^\top h)\,\hat{v}$, the component of h lying along $\hat{v}$, and discards everything else. **Ablation**, as throughout this book, means removing a candidate component of the computation to observe what changes. **Projection ablation** composes the two:

$$h \;\leftarrow\; (I - \hat{v}\hat{v}^\top)\,h \;=\; h - (\hat{v}^\top h)\,\hat{v},$$

subtract from h exactly its own component along $\hat{v}$. The contrast with addition is why the operation is needed at all. Adding $-\alpha\hat{v}$ pushes every input by the same fixed amount: overshooting where the concept is already weak, undershooting where it is strong. Projection ablation is input-adaptive: it removes precisely as much of the concept as the activation contains, guarantees a zero reading along $\hat{v}$ for every input, requires no dose parameter α, and is idempotent (applying it twice changes nothing further). It is the tool of choice when the goal is *removal* rather than steering - the refusal-direction ablation of Section 7.1 is exactly this operation.

Both are special cases of an *affine* map

$$h \;\leftarrow\; Wh + b,$$

addition being $W{=}I$, $b{=}\alpha\hat{v}$ and projection ablation $W = I - \hat{v}\hat{v}^\top$, $b{=}0$. The family between them is large, but it is worth being precise about what even the full affine family, used one direction at a time, does not buy:

- **Linear component only.** A projection removes the linearly encoded part of a concept. Anything encoded nonlinearly, or distributed across several correlated directions, passes through untouched - the circular and conditional features of Section 5.6 are the standing examples.
- **A line is a crude summary of a region.** A single direction is a one-dimensional statement, but real attributes (register, sentiment, topic) occupy *regions* of activation space with variance along many directions at once.
- **Composition interferes.** Concept directions are correlated rather than orthogonal (superposition guarantees it, §5.4), so stacking single-direction edits compounds side effects: three nudges that each work alone need not work together.

Two lines of work address these limits with principled structure - one from recurrent-network dynamics, one from optimization theory.

7.2.2 Conceptors: soft projections with a Boolean algebra

The first line answers the region and composition limits directly. **Conceptors** (Jaeger, 2014), developed for reservoir and recurrent networks, capture the *region* of state space a network occupies while doing a particular task. From the state correlation matrix $R = \mathbb{E}[hh^\top]$, the conceptor is

$$C \;=\; R\,(R + \alpha^{-2} I)^{-1},$$

a soft projection whose eigenvalues lie in $[0, 1)$: directions with high task variance are preserved (eigenvalue $\to 1$), low-variance directions suppressed, with **aperture** α setting the softness. Where a steering vector says "move along this line," a conceptor says "stay in this ellipsoid."

The distinctive property is compositionality. Conceptors support a Boolean algebra with closed-form matrix expressions (Jaeger, 2014): AND (intersection of ellipsoids), OR (union), NOT (complement). The original paper demonstrated a reservoir network storing dozens of temporal patterns, each gated by its conceptor, with logical combinations morphing smoothly between patterns. For transformer steering this is the natural formalism for the question additive vectors answer badly: how do you steer toward *formal* AND *honest* but NOT *verbose* without three nudges interfering? In early evaluations, conceptor-based steering on language-model activations has outperformed additive vectors on multi-concept control, though systematic comparisons remain young; the cost is a $d \times d$ matrix per concept instead of a d-vector.

7.2.3 Affine steering with guarantees

The second line answers a different gap: among all affine maps, which one? Singh, Ravfogel, Herzig, and colleagues (2024), in *Representation Surgery*, derive rather than propose the intervention: state a formal objective and solve for the optimal affine map. Two results anchor the paper. For **concept erasure**, making the steered representations carry no linearly decodable signal about a concept ("guardedness"), the minimally distorting affine intervention has a closed form, in the lineage of LEACE (Belrose et al., 2023): an oblique projection removing exactly the concept subspace, whitened so distortion is minimal. For **steering toward a class**, matching the target class mean is optimal under their objective, and matching covariances (a full affine map, not a translation) provably reduces the residual signal further.

The practical import is less any single formula than the workflow it licenses: *say what you want the intervention to achieve, in math, and derive the operator* - replacing the guess-a-vector-and-sweep-α loop with statements you can check. When the derived optimum is affine, you also learn something about the ceiling: if provably-optimal affine erasure still leaks the concept to a nonlinear probe, the leak measures exactly how non-linear the concept's encoding is, connecting back to the exceptions to the linear representation hypothesis in Section 5.6.

Neuroscience parallel. Control as shaping a region of state space, rather than pushing along a single axis, is the modern picture of motor cortex. Churchland, Shenoy, and colleagues (2012) recast movement preparation as setting an *initial condition inside a subspace*, with movement unfolding as the lawful dynamical evolution from it; follow-up work distinguished "output-null" directions (along which activity can vary without affecting muscles) from "output-potent" directions (which drive output). An affine steering operator formalizes exactly that distinction for transformers: edit designated subspaces, guard the rest.

7.3 Practical Comparisons: Prompting vs. Fine-tuning vs. Steering

7.3.1 The three levers

Every behavior change you want from a model is implemented through one of three levers, and choosing is an engineering decision with measurable trade-offs - not a matter of taste.

	Prompting	**Fine-tuning**	**Steering**
Marginal cost	context tokens each call	GPU-hours once; serving copies	one forward hook; ∼zero latency
Reversibility	total - edit the string	poor - retrain or revert weights	total - remove the hook
Precision	coarse; competes with rest of context	high where data covers; drifts elsewhere	surgical *iff* concept is linear
Durability vs. user pressure	weak - injectable, exhaustible	strong	strong (user can't reach activations)
Side-effect profile	prompt crowding, injection surface	broad, delayed, hard to audit	measurable, dose-controlled
Auditability	full (text is the artifact)	low (weight diff is opaque)	medium (vector + α + eval)
Fails when	context is adversarial or long	data underspecifies intent	concept isn't linearly encoded

7.3.2 What each lever is actually for

Prompting is the default for a reason: zero infrastructure, instant iteration, complete reversibility. Its two structural weaknesses are that it occupies the same channel as attacker input (prompt injection is prompting, hostile) and that instructions decay over long contexts. System-prompt behavior is a suggestion the model usually honors, not a constraint.

Fine-tuning (supervised fine-tuning, LoRA, RLHF) is the durability lever: it changes what the model *is* rather than what it is told, generalizes beyond demonstrated cases, and survives adversarial users. Its costs are symmetric: changes generalize in unintended directions too. Emergent misalignment (Betley et al., 2025), where fine-tuning on insecure code produces broadly misaligned personas, is the standing cautionary tale: the fine-tuning data said one narrow thing, and the optimizer learned a broad latent trait, invisibly, discovered only by later behavioral audit. Fine-tuning's side effects are broad, delayed, and unauditable by inspection - you find them by evaluation or not at all. (Part I, §4.4, covers the emerging repair: concept-ablation fine-tuning projects out unwanted activation directions *during* training and largely prevents the effect; Casademunt et al., 2025.)

Steering occupies the middle: more durable than prompting against user pressure (the user cannot edit activations), more reversible and auditable than fine-tuning (the intervention is a vector you can print, dose, and delete), and surgical exactly when Chapter 5's geometry cooperates. Its boundary is the boundary of the linear representation hypothesis: behaviors without a clean linear encoding steer badly, and multi-concept steering inherits the interference problems Section 7.2 exists to solve.

7.3.3 Composition is the real answer

Production systems increasingly use all three, assigned by the table's rows: fine-tune what must be durable and general (core safety, format compliance); steer what must be adjustable, reversible, and user-tamper-proof (per-deployment persona, monitored-concept suppression); prompt what changes per-request. Monitoring completes the loop: probes (Section 7.1's reading vectors) detect a concept cheaply at serving time, conditional steering responds only when triggered, and fine-tuning absorbs whatever the monitoring logs prove is a persistent gap. The three levers compose into a control system - interpretability supplying the sensors, steering the actuators, and fine-tuning the slow adaptation layer.

A fourth lever is emerging that spans the table's rows: **training-time steering**, surveyed in Part I (§4.4). Gradient routing localizes a capability into a designated subregion during training so it can later be removed cleanly (Cloud et al., 2024); concept-ablation fine-tuning steers *which* solution the optimizer selects from the data (Casademunt et al., 2025); preventative steering adds a trait direction during fine-tuning so the model absorbs training pressure without encoding the trait (Chen et al., 2025). These methods have fine-tuning's durability with steering's precision and auditability - the same interpretability instruments this chapter uses at inference time, moved inside the training loop. They are young, validated mostly on small models and fine-tuning rather than pretraining at scale, and they belong in any modern version of the comparison above.

Takeaways

- Contrast-pair steering vectors change behavior with no training (Turner et al., 2023; Zou et al., 2023); layer and dose are the knobs, and dose–response with capability retention is the required evaluation. The refusal direction, one vector mediating refusal across 13 models (Arditi et al., 2024), is the area's central causal result.
- Operators generalize vectors: conceptors (Jaeger, 2014) contribute soft projections with a Boolean algebra for compositional control; affine steering theory (Singh et al., 2024) contributes provably optimal interventions for stated objectives - and residual leakage under optimal affine erasure measures a concept's non-linearity.
- Prompting, fine-tuning, and steering trade off cost, reversibility, precision, durability, and auditability; mature systems compose all three, with probes as sensors and steering as the fast reversible actuator.

Research Directions

From §7.1: Conditional steering for prompt injection

Always-on steering taxes every token; conditional steering applies the vector only when a probe fires. Prompt injection is the natural target: train a probe for "instructions arriving from retrieved/tool content rather than the user," and fire a compliance-suppressing vector only on those spans. Deliverables: injection-success rate against a standard attack suite, benign-task degradation (which should be ≈ 0 when the probe is silent), probe false-positive rate at

deployment thresholds, and latency overhead. Every component is standard; the composition (and a candid side-effect accounting) is the contribution.

From §7.2: Operator steering against eval awareness

Models increasingly behave differently when they detect evaluation contexts - a direct threat to every benchmark-based safety claim. Suppressing eval awareness with a crude additive vector risks collateral damage to situational reasoning in general. The sharper tool: a conceptor or guarded affine map that removes *evaluation-context* information while explicitly preserving a protected subspace for adjacent capabilities (instruction-following, context-tracking). Deliverables: does behavior-under-evaluation converge to behavior in deployment-like contexts, what breaks, and (the governance question) should results obtained under suppression be trusted more than results without it, or does the intervention itself invalidate the eval?

From §7.3: A benchmark for behavior-change interventions

No public benchmark compares the three levers on equal footing. Fix a suite of target behaviors spanning difficulty (output format $\ll$ persona $\ll$ suppressing a trained behavior); implement each via strong versions of all three levers; measure on a shared grid - success rate, capability retention, robustness to adversarial pressure, off-target drift, latency, and *durability decay* over context length and multi-turn conversations. The deliverable is the trade-off table of Section 7.3 with numbers in the cells, per behavior class. Every deployment team is currently building a private, partial version of this; the first public one becomes the field's reference.

Programs for Chapter 7

`refusal_direction_steering.py` On a small chat model (Qwen-1.8B-class): compute the refusal direction from 100 harmful/harmless instruction pairs at a middle layer. Implement both interventions: projection ablation ($h \leftarrow h - \hat{v}\hat{v}^\top h$ at every layer) and addition ($+\alpha\hat{v}$). Measure refusal rate on 50 held-out harmful and 50 harmless prompts, and score a 200-question MMLU subset under each condition. Plot the full dose–response for five values of α. *The point of the exercise is the second plot* - behavior change means nothing until you price its capability tax. *Type: script + matplotlib. Deps: torch, transformers.*

`conceptor_steering.py` Two parts. (1) Classic: a 500-unit echo state network driven by four periodic patterns; compute each pattern's conceptor, then regenerate patterns by gating with C, and morph by interpolating conceptors - the demonstration from Jaeger (2014) in ∼150 lines of numpy. (2) Transformer: compute conceptors for *positive* and *formal* sentiment/register from contrast sets of GPT-2 activations; steer with AND($C_{\text{pos}}, C_{\text{formal}}$) vs. the sum of two additive vectors; score continuations for both attributes. The head-to-head on *simultaneous* two-attribute control is where operators visibly justify their extra cost. *Type: numpy + torch script.*

`prompt_finetune_steer_comparison.py` One target behavior: *always respond in valid JSON with keys `answer` and `confidence`.* Three implementations on the same small chat model: (a) system prompt; (b) LoRA fine-tune on 500 synthetic examples; (c) steering vector from JSON-compliant vs. free-text contrast pairs. Score each on: compliance rate over

200 held-out queries, compliance after an adversarial user turn ("ignore the format, just chat with me"), MMLU-subset retention, and tokens/sec overhead. Print the results as the trade-off table from Section 7.3, with your measured numbers in the cells - the entire chapter, empirically, in one afternoon. *Type: script; optional* `peft` *for LoRA.*

8. Evaluation and Frontiers

Chapters 5–7 produced tools and stories: geometric measurements, feature dictionaries, circuit diagrams, steering vectors. This chapter asks the two questions that determine whether any of it matters. First: *when should we believe an interpretation?* Second: *what should the field work on next?*

Structure of this chapter. Section 8.1 examines the **faithfulness problem** - the field's central unsolved challenge: explanations can be plausible, predictive, and wrong, and the failure is usually silent. Section 8.2 surveys the search for a quantitative theory underneath the descriptions, where **information geometry** and the study of training dynamics (grokking, phase transitions, discrete skill acquisition) suggest that internal reorganization precedes behavioral change. Section 8.3 lays out the *open problems*: scaling current methods to frontier and mixture-of-experts models, closing the verification gap, automating the analysis loop, and the field's turn toward applied questions, ending with a ranked starter map for the reader. If you read only one chapter of this part closely, this is the one to choose: the most important content of interpretability is not any particular finding but the standard of evidence, and that standard is still under construction.

8.1 The Faithfulness Problem

8.1.1 Plausible is not faithful

An explanation of a model's behavior is **plausible** if it makes sense to a human and fits the observations examined. It is **faithful** if it describes the computation the model actually performed. The gap between these is where interpretability fails silently: a plausible story constrains nothing, because for any behavior there are many plausible stories, and the incentives of the explainer (human or model) push toward persuasiveness, not truth.

8.1.2 The microprocessor cautionary tale

The sharpest demonstration comes from outside machine learning. Jonas and Kording (2017) asked: if we apply standard neuroscience analysis methods to a system whose mechanism we *fully understand* (the MOS 6502 microprocessor running Donkey Kong), do the methods recover the truth? They lesioned every one of the chip's 3,510 transistors individually and identified the subset whose removal breaks a specific game: transistors a neuroscientist would label "Donkey Kong transistors." The label is mechanistically meaningless: those transistors implement generic logic used by many programs, and the lesion deficit reflects incidental dependency structure, not function. Spike-train-style analyses of transistor switching, tuning curves, and dimensionality reduction on chip activity likewise produced familiar-looking, publishable, *wrong* characterizations. Methods validated by decades of neuroscience practice failed to recover the mechanism of a system orders of magnitude simpler than any brain - and the failure was only visible because ground truth was available (Jonas & Kording, 2017).

The transfer to interpretability is direct and uncomfortable: ablate an attention head, observe a behavioral deficit, and you have exactly the "Donkey Kong transistor" evidence pattern. The backup name-mover heads of Section 6.2 are the same lesson from inside the field. A deficit under lesioning is weak evidence about what the lesioned component was computing; the paper is the standing argument for the verification ladder of Section 6.3, and for building interpretability's own known-ground-truth test systems (Section 8.3).

8.1.3 Chain-of-thought faithfulness: the live instance

The faithfulness problem is most consequential where the model explains *itself*. Reasoning models emit chains of thought, and an appealing safety strategy is to read them: if the model plans something bad, the plan should be visible. Whether that works is an empirical question, and the results so far are mixed in both directions.

Lanham et al. (2023) introduced the intervention paradigm: perturb the chain of thought and see if the answer cares. **Truncate** the reasoning mid-stream, **corrupt** a step with an introduced error, **paraphrase** it, or replace it with **filler tokens**: if the final answer is unchanged, the stated reasoning played no causal role in producing the answer. They found wide variation: on some tasks answers track the chain of thought tightly; on others the model reaches the same answer with its reasoning removed, meaning the stated reasoning was post-hoc narrative. Faithfulness also varied non-monotonically with model size - larger models can be *less* faithful on easy tasks, plausibly because they don't need the scratchpad (Lanham et al., 2023).

Arcuschin et al. (2025) showed that unfaithfulness arises *naturally*, without adversarial pressure: models exhibit implicit post-hoc rationalization (for example, answering "yes" to both *"is X bigger than Y?"* and *"is Y bigger than X?"* with fluent, contradictory justifications), along with silent error-correction ("illusory transparency"), where the printed reasoning contains mistakes that the underlying computation silently repairs. Bogdan, Macar, Nanda, and Conmy (2025) supply the constructive response with *thought anchors*: sentence-level resampling and attention suppression identify *which* sentences of a chain of thought actually influence the final answer, recovering a dependency graph of the reasoning rather than trusting its surface.

For monitoring, the operational summary is: a chain of thought is evidence, not testimony - a channel that correlates with the underlying computation, sometimes strongly, and can be audited by intervention, but never read as ground truth. Every monitoring proposal built on reading chains of thought needs an answer to "what is your measured faithfulness rate on this model and task family, at what false-positive budget?"

8.1.4 A catalog of interpretability illusions

Recurring ways an interpretability result can be true-looking and false, collected from Chapters 5–7:

- **Probe illusion** (§5.6): the probe reads a direction that is present but unused downstream. Antidote: intervention.
- **Lesion illusion** (§6.3; Jonas & Kording, 2017): deficit under ablation read as function. Antidote: patching with targeted counterfactuals; multiple ablation references.
- **Off-distribution illusion** (§6.3): interventions (zero ablation, noised embeddings) push the model off-distribution, and the measured effect is the distribution shift, not the mechanism.

- **Resolution illusion** (§6.1): sparse-autoencoder feature sets depend on dictionary size; a "discovered feature" may be an artifact of where you stopped splitting.
- **Narrative illusion** (this section): the explainer (researcher or model) optimizes for coherence. Chain-of-thought rationalization is the model-side case; cherry-picked feature dashboards are the researcher-side case.

Neuroscience parallel. Every illusion above was catalogued in neuroscience first, under its own name: reverse inference (decodability mistaken for use), lesion-deficit confounds and post-lesion compensation, stimulation artifacts, atlas-resolution dependence, and the file drawer of pretty-but-wrong tuning-curve stories. Neuroscience built its methodological immune system (preregistration, causal-inference standards, model organisms with known circuits) after decades of these failures. Interpretability, moving faster on cheaper experiments, can import that immune system instead of re-evolving it; Jonas and Kording (2017) is the transfer document.

8.2 Information Geometry Meets Interpretability

8.2.1 Why look for a deeper theory

Everything in Chapters 5–7 is descriptive: measured profiles, discovered features, documented circuits. Nothing yet *predicts* - when a capability will emerge, where a feature will form, which training run will generalize. A growing research thread reaches for mathematical structure underneath the descriptions, borrowing from information geometry, statistical physics, and occasionally further afield. This section surveys the thread with its epistemic status plainly marked: promising reframings, few established results.

8.2.2 The loss landscape as a geometric object

Part I built this geometry in full (its Chapters 2 and 4): the **Fisher information matrix** is the natural metric on parameter space, measuring how much the model's output distribution changes per unit parameter change (Amari, 1998), and training navigates the manifold it defines. What this section adds is the interpretability reading. The object of interest is the *spectrum*: flat directions of F are parameters the behavior doesn't care about; sharp directions are the parameters the model's behavior depends on most strongly. Flatness connects (with the reparameterization caveats Part I treats) to generalization and robustness of the learned representation, and Fisher-weighted parameter importance is what continual-learning methods compute when deciding what not to overwrite (Kirkpatrick et al., 2017) - a bridge between geometry and "which weights encode the capability."

8.2.3 Training dynamics as trajectories with transitions

Two results anchor the claim that training has *structure* (discrete events, not smooth drift) and that internal reorganization precedes behavioral change.

The first is **grokking** (Power, Burda, Edwards, Babuschkin, and Misra, 2022): small transformers trained on modular arithmetic reach 100% *training* accuracy quickly, then sit at chance *test* accuracy for orders of magnitude more optimization steps (10^2 to 10^5 steps of apparent stasis) before test accuracy jumps abruptly to near-100%. Follow-up mechanistic analysis (Nanda et al., 2023) showed what the plateau hides: the network is gradually assembling a generalizing

algorithm (for modular addition, a Fourier-transform-based circuit using trigonometric identities) while the memorizing solution still dominates behavior; the "sudden" generalization is the handover. (Part I, §4.4, examines the training-side lever: weight decay drives the cleanup phase that completes it.)

A caveat worth stating plainly, because it is easy to misread grokking as a goal: *generalization* is the objective, and reaching it *immediately* is the better outcome in practice. Grokking is not a superior end state; it is the same end state reached by a slow, delayed route. Its value is diagnostic. Grokking is a *lab curiosity*: it requires a specific regime (a small, over-parameterized model, a *tiny* training fraction, and strong weight decay) engineered so that memorization is the cheapest path to zero training loss and therefore wins first. Because the general circuit then forms only slowly, under weight-decay pressure, over tens of thousands of steps, the long plateau lets us *watch the circuit assemble* rather than have it appear instantly and invisibly. Widen the training fraction (or add data) and the rule becomes the cheapest path from the start: the network generalizes at once, and there is no delay to instrument. The phenomenon is thus a property of the *training regime*, not of modular arithmetic - a controlled setting for observing circuit formation, not a behavior one wants in a deployed model.

The second is the **induction-head phase change** of Section 6.2 (Olsson et al., 2022): a loss-curve bump, circuit formation, and in-context learning arriving together in a narrow token window across model sizes.

The interpretability reading: behavioral metrics are lagging indicators. Circuits form first; capabilities follow. That inverts the evaluation problem - if internal observables (Fisher spectra, weight-space structure, feature-dictionary drift) can be monitored *during* training, emergence stops being a surprise found at benchmark time. This is not left as a promise: the chapter's `fisher_spectrum_inspector.py` program tracks the top Fisher eigenvalues through the grokking run itself, and in the book's reference run the largest spectral movement precedes the test-accuracy jump by $\sim$400 steps - a spectral crisis (a $\sim 10\times$ eigenvalue spike as the memorized solution destabilizes) arriving while behavior still reads as chance. Two internal observables, two early-warning profiles: the embedding Fourier share moves earlier but gradually; the Fisher spectrum moves later but violently.

Michaud, Liu, Girit, and Tegmark (2023) extend the picture to scale with the **quantization model of neural scaling**: model the world as a long-tailed (Zipf-distributed) collection of discrete prediction "quanta" (skills, facts, patterns) that models acquire in rough order of usefulness. Summing the per-quantum loss reductions over the tail yields power-law scaling laws, and predicts that smooth aggregate loss curves decompose into many small discrete acquisitions - consistent with what they find by clustering per-token loss trajectories. Smoothness at the macroscale, mechanism at the microscale: the same claim grokking makes about time, made about scale.

On the **quantum-geometric analogies** that occasionally appear in this literature: there is emerging work importing formalism from quantum geometry (metric structures on state spaces, geometric phases) as analogy for representation dynamics. As of this writing it is speculative - no established interpretability result depends on it. The fair summary: the demonstrated value of geometry-of-training is real (Fisher structure, phase transitions, quanta); the exotic extensions are watchlist items, cited here so the reader recognizes them, not so the reader believes them.

Neuroscience parallel. Neuroscience arrived at the same two commitments, geometry and transitions, from its own data. Neural population activity during learning traces low-dimensional

manifolds that *reorganize* at behavioral milestones; developmental neuroscience's critical periods are phase transitions in plasticity; and the analysis vocabulary (state-space trajectories, attractor landscapes, dimensionality collapse at learning onset) transfers to training-dynamics work almost without translation. The shared bet is that learning systems of any substrate reorganize through discrete geometric events - and each field now supplies test cases for the other's theories.

8.3 Open Problems and Future Tools

8.3.1 The scaling wall

The methods of Chapters 5–7 were validated mostly on models between 70M and 8B parameters. Frontier models are 100–1000× larger, and three gaps open with scale. **Cost**: sparse-autoencoder training at the 16-million-latent scale already consumes tens of billions of tokens of activation data per layer (Gao et al., 2024); covering every layer of a frontier model at adequate dictionary resolution is a pretraining-class expenditure, and circuit analysis scales worse. **Architecture**: mixture-of-experts models add a discrete routing layer that current tools barely address - features and circuits are conditional on routing paths, and the combinatorics of path-conditioned analysis is untouched territory. **Modality**: multimodal models fuse representations across encoders; whether concept directions and feature dictionaries transfer or fragment across the fusion boundary is mostly unmeasured.

8.3.2 The verification gap

There is no agreed standard for "this explanation is correct": Section 6.3's ladder is practice, not benchmark. The missing infrastructure is **known-mechanism model organisms**: systems where ground truth exists by construction (hand-compiled transformers, models trained on algorithmic tasks with verified circuits, deliberately backdoored models), against which verification methods themselves can be scored for sensitivity and specificity - the interpretability analogue of the Jonas–Kording chip, built on purpose. The auditing-game format (hide a behavior in a model; score teams and tools on finding it) is the most promising current instantiation, and it doubles as the natural benchmark for the next item.

8.3.3 Automation and the applied turn

Interpretability labor is expert-bound, and the field's response is to spend model capability on model transparency: language-model agents generating hypotheses, choosing prompts, running patching sweeps, auto-labeling sparse-autoencoder features, and, in auditing settings, discovering implanted behaviors end-to-end. The open question is whether automated interpretability *keeps pace* with the capabilities of the models it must explain: transparency per FLOP has to grow at least as fast as capability per FLOP, or the gap between what models can do and what we can explain never closes.

Meanwhile the field's center of gravity has moved from pure reverse-engineering toward **model biology**, the stance Part I introduced in its Section 3.5: treating models as organisms whose behaviors are studied with behavioral evidence first and causal evidence second, without demanding complete mechanistic reduction before saying anything; the behaviors in question include emergent misalignment (Betley et al., 2025), eval awareness, sycophancy, and user modeling. The applied portfolio is concrete: probe-based monitoring in production (cheap,

and currently the state of the art for misuse detection), model diffing after fine-tuning (what changed?), forensics after incidents (why did it do that?), and steering as a deployment control (Section 7.3). These are the tasks where interpretability either delivers measurable value or doesn't - recall at fixed false-positive budgets, incident-explanation accuracy on held-out counterfactuals.

8.3.4 Tools that do not exist yet

A short list of buildable-but-unbuilt infrastructure, offered as an agenda:

- **A causal-claim verifier**: given a mechanism hypothesis in a standard format, automatically compile it to the cheapest patching/scrubbing experiment set that could reject it, run, and report.
- **Cross-model feature dictionaries**: sparse-autoencoder features aligned across model families and scales, so "the sycophancy feature" becomes a portable, versioned object rather than a per-model rediscovery.
- **Interpretability foundation models**: models trained to read activations (natural-language autoencoders, activation-to-explanation translators), amortizing the per-question cost of internal analysis.
- **Training-time monitors**: the leading-indicator program of Section 8.2 productized - dashboards over Fisher spectra, feature-formation events, and loss-quanta acquisition during pretraining.
- **A verification benchmark**: the known-mechanism organism suite above, with public leaderboards for verification methods, not just discovery methods.
- **Training-time levers at scale**: the train-for-legibility and train-for-steerability programs surveyed in Part I (§4.4), which include gradient routing (Cloud et al., 2024), concept-ablation fine-tuning (Casademunt et al., 2025), and preventative steering (Chen et al., 2025), are validated so far on small models and fine-tuning. Nobody has trained a frontier-scale model with interpretability constraints from the start, or priced the interpretability tax at scale. If the price is affordable, most of this chapter's post-hoc machinery becomes a verification layer rather than the primary tool.

8.3.5 A starter map for the reader

Ranked by (ease of entry × impact), for a reader with this book and a laptop - each item traces to a Research Direction or Program in one of the chapters:

1. **Chain-of-thought faithfulness measurement** (§8.1): behavioral, cheap, directly safety-relevant; `chain_of_thought_faithfulness.py` is the on-ramp.
2. **Probe-based monitoring at fixed false-positive rates** (§5.6, §7.3): the applied task where linear tools are already state of the art.
3. **Grokking-style training-dynamics studies** (§8.2): fully reproducible at toy scale; the leading-indicator question is open.

4. **Steering side-effect accounting** (§7.1–7.2): every new steering method needs the dose–response-plus-capability evaluation; supplying it for published methods is publishable.
5. **Sparse-autoencoder evaluation against cheap baselines** (§6.1): negative results welcome and needed.
6. **Model diffing and forensics** (§8.3): highest realism, needs the most judgment; start after two or three of the above.

Takeaways

- Plausible ≠ faithful, and the failure is silent: the microprocessor study (Jonas & Kording, 2017) shows standard methods produce confident wrong stories when ground truth is checkable; chain-of-thought unfaithfulness (Lanham et al., 2023; Arcuschin et al., 2025) is the same phenomenon where the model narrates itself. Intervention is the only antidote, and it has its own illusions to manage.
- Training has geometric structure: phase transitions (Power et al., 2022; Olsson et al., 2022), Fisher-metric landscapes, and discrete skill quanta (Michaud et al., 2023) - circuits form before capabilities appear, making internal observables potential leading indicators. The exotic geometric analogies remain speculative, and are labeled as such here.
- The frontier is scaling (cost, mixture-of-experts, multimodal), verification (known-mechanism organisms, claim-checking infrastructure), automation (transparency per FLOP vs. capability per FLOP), and the applied turn (monitoring, diffing, forensics) - where interpretability's value is finally measured in deployment terms.

Research Directions

From §8.1: Chain-of-thought faithfulness monitors with explicit error budgets

Build a monitor that reads reasoning transcripts and flags suspicious content, then (the step almost no one does) *calibrate it against intervention ground truth*: on a task set where truncation/corruption tests establish which chains of thought actually determine the answer, measure the monitor's recall at fixed false-positive rates (0.1%, 1%). A monitor that fires on 30% of benign transcripts is unusable at scale; one with unknown recall on unfaithful reasoning is unfalsifiable. The deliverable is the ROC curve nobody has published, per model family - and if faithfulness varies more by training recipe than by scale, as early results hint, the curves will differ across models in interesting ways.

From §8.2: Predict circuit formation before behavior shows it

Instrument a small-model training run (modular arithmetic for grokking, or a 2–4 layer language model for induction heads) with cheap per-checkpoint observables: Fisher/Hessian top spectrum, embedding Fourier energy, weight-matrix effective rank, per-token-cluster loss (the quantization lens). Question: which observable moves *earliest* before the behavioral transition, and how reliable is it as a leading indicator across seeds? A validated early-warning signal for

capability emergence, even in toy settings, would be the first *predictive* artifact interpretability has produced, and the protocol scales conceptually to monitoring real pretraining runs.

From §8.3: Build one known-mechanism organism

Train a small transformer on an algorithmic task until its circuit is fully verified (or hand-compile one), then deliberately write three plausible-but-wrong mechanism stories for it. Score each verification method from Section 6.3 (probing, three ablation flavors, patching configurations, scrubbing) on whether it rejects the wrong stories and retains the right one, at what compute cost. Publish the organism, the stories, and the scores as a first verification benchmark.

Programs for Chapter 8

`chain_of_thought_faithfulness.py` Hint-injection on a small reasoning-capable model. Take 200 multiple-choice questions; present each twice: clean, and with an embedded hint ("a Stanford professor thinks the answer is (B)"). Measure: (1) P(answer changes | hint); (2) among changed answers, P(reasoning mentions the hint). The gap between influenced-by and acknowledged-influence is the unfaithfulness rate. *Extension:* add truncation - cut the chain of thought at 25/50/75% and force the answer; plot answer stability vs. truncation point to measure how much of the reasoning actually determined the answer. *Type: script + matplotlib. Deps: torch, transformers.*

`grokking_phase_transition_for_modular_arithmetic.py` Train a 2-layer transformer (width 128, 4 heads; the original Power et al. architecture size, $\sim 4 \times 10^5$ non-embedding parameters) on $a+b$ mod 97 with weight decay on, $\sim 10^5$ steps (minutes on a laptop GPU, hours on CPU). The `--train-frac` knob sets the regime: the default 0.3 holds out most pairs and sits in the grokking window, whereas 0.5 gives enough coverage that the network generalizes immediately with no delay to observe. Log train/test accuracy, weight norm, and the payoff: the Fourier power spectrum of the token embedding matrix every 500 steps. You will watch: train accuracy saturate, a long nothing, sparse Fourier peaks growing quietly, then test accuracy snapping upward as the peaks sharpen. Grokking, circuit formation, and internal-change-before-behavioral-change, on one screen. *This is the book's single best experiment. Type: training script + matplotlib.*

`interpretability_case_study.py` (capstone) The full loop, once, end-to-end. Pick a single odd behavior of a 1–7B open model (a systematic factual error, a format quirk, an inconsistent refusal). Then: (1) *quantify*: prevalence across 200+ prompt variants, seeds, temperatures, with confidence intervals; (2) *hypothesize*: write down two competing causal stories before touching internals; (3) *probe*: one linear probe or logit-lens scan positioned to discriminate the stories; (4) *intervene*: one targeted patch, ablation, or steering test per story; (5) *report*: a two-page verdict with effect sizes, the experiment that would overturn it, and what remains unexplained. Steps 2 and 5 are the ones that make it science. *Type: project scaffold; all prior programs are its parts.*

`fisher_spectrum_inspector.py` Section 8.2's claim - internal observables reorganize before behavior changes - put to an actual test. Re-run the grokking experiment and, every few hundred steps, estimate the top eigenvalues of the *true Fisher information matrix*

by the methods of Part I §2.2: Fisher–vector products (never the matrix; exact vector-Jacobian products through a retained graph, finite-difference directional derivatives for the forward side) fed to a Lanczos iteration. Plot the spectrum's trajectory against test accuracy and the Fourier order parameter, and print the lead or lag between the largest spectral movement and the behavioral jump. The result is reported as a measurement, not asserted in advance: a spectrum that moves first supports §8.2's monitoring program; one that moves late bounds it. Reference run (seed 0): the top eigenvalue climbs three orders of magnitude through the plateau, spikes $\sim 10\times$ at step ~ 4800 while train accuracy briefly buckles - the memorized solution destabilizing - then collapses as test accuracy snaps up; the largest spectral movement (step 5200) *leads* the behavioral jump (step 5600) by ~ 400 steps. Replicate this and the grokking run together with `run_grok.sh`. *Type: training script + matplotlib; CPU-tractable at* 4×10^5 *parameters.*

A. A Timeline of Interpretability and Model Biology Research, 2019–2026

The results discussed in Chapters 5–8 arrived in a particular order, and the order itself tells a story: from probing single networks (2019), through circuit-level reverse engineering (2020–2022), the sparse-autoencoder wave (2023–2024), the "model biology" era of behavioral and applied work (2024–2025), and into the newest phase, where control moves into the training loop itself (2024–2026). Dates are publication dates of the cited papers or posts; where a month is well established it is given.

2019 - Geometry and probing

- Ansuini, Laio, Macke, and Zoccolan measure intrinsic dimension across layers of trained vision networks and find the rise-then-fall "hunchback" profile, with final-layer compression predicting generalization (NeurIPS 2019). [§5.3]
- The BERTology wave: probing studies show syntactic and semantic properties are linearly decodable from transformer layers, ordered roughly like a classical NLP pipeline (Tenney et al., 2019). The correlational-vs-causal gap in probing becomes a recognized methodological concern (Hewitt & Liang, 2019). [§5.6]
- Barrett, Morcos, and Macke publish the survey connecting analysis methods for biological and artificial networks - the two-way methodological bridge this book's neuroscience parallels draw on.

2020 - Circuits on vision models

- Olah et al. publish *Zoom In: An Introduction to Circuits* (Distill, March 2020), proposing that networks decompose into features connected by weights into circuits - the founding manifesto of mechanistic interpretability. Curve detectors, high-low frequency detectors, and other InceptionV1 circuits are documented through the year. [§6.2]
- The logit lens (nostalgebraist, August 2020) shows intermediate transformer layers can be read through the unembedding - an early, cheap window into layerwise computation.

2021 - Transformers get a framework

- Anthropic's interpretability team publishes *A Mathematical Framework for Transformer Circuits* (Elhage et al., December 2021): residual stream as shared communication channel, attention heads as read-write operations, QK/OV decomposition - the vocabulary in which Chapters 5–7 are written.

- Power, Burda, Edwards, Babuschkin, and Misra post the grokking paper (January 2021 workshop; arXiv January 2022): delayed generalization on algorithmic tasks, showing training dynamics have structure that behavioral metrics miss. [§8.2] (Listed here under first appearance; cited as Power et al., 2022, elsewhere in this book.)

2022 - The circuit era peaks

- Olsson et al. document induction heads and their phase change during training, tying a specific circuit to in-context learning (March 2022). [§6.2]
- Elhage et al. publish *Toy Models of Superposition* (September 2022): sparsity-driven phase transitions, polygon packing geometries, and the argument that polysemanticity is rational capacity allocation. [§5.4]
- Wang, Variengien, Conmy, Shlegeris, and Steinhardt reverse-engineer the indirect-object-identification circuit in GPT-2 small - 26 heads, 7 classes, backup heads, and the faithfulness/completeness/minimality criteria (November 2022). [§6.2]
- Chan et al. propose causal scrubbing: testing a complete mechanism hypothesis by resampling everything it claims is irrelevant (December 2022). [§6.3]
- Meng et al.'s ROME work shows factual associations can be located and edited in mid-layer MLPs - early evidence that knowledge is spatially organized enough to intervene on.

2023 - Sparse autoencoders and steering

- Turner et al. introduce activation addition: steering with contrast-pair difference vectors, no optimization (August 2023). [§7.1]
- Cunningham, Ewart, Riggs, Huben, and Sharkey show sparse autoencoders recover highly interpretable features from language-model activations (September 2023); Anthropic's *Towards Monosemanticity* (Bricken et al., October 2023) demonstrates the same at greater depth on a 1-layer model. The dictionary-learning era begins. [§6.1]
- Zou et al. publish *Representation Engineering* (October 2023): reading vectors and control vectors as a general top-down transparency paradigm. [§7.1]
- Zhang and Nanda systematize activation patching (metrics, corruption choices, denoising vs. noising), turning a folk method into a documented one (September 2023). [§6.3]
- Park, Choe, and Veitch formalize the linear representation hypothesis and the causal inner product (November 2023). [§5.6]
- Lanham et al. publish the chain-of-thought faithfulness testing paradigm: truncation, corruption, paraphrase, filler tokens (July 2023). [§8.1]
- Michaud, Liu, Girit, and Tegmark propose the quantization model of neural scaling: discrete skill quanta under smooth loss curves (NeurIPS 2023). [§8.2]

2024 - Scale, and the applied turn begins

- Singh, Ravfogel, et al. publish *Representation Surgery*: provably optimal affine steering and erasure (February 2024). [§7.2]
- Anthropic's *Scaling Monosemanticity* (Templeton et al., May 2024) trains sparse autoencoders on Claude 3 Sonnet, finds safety-relevant features, and demonstrates feature steering publicly ("Golden Gate Claude"). [§6.1]
- Gao et al. (OpenAI) train a 16-million-latent TopK sparse autoencoder on GPT-4 activations and publish scaling laws and an evaluation suite for dictionaries (June 2024). [§6.1]
- Arditi et al. show refusal is mediated by a single direction across 13 open chat models, with weight orthogonalization as a training-free jailbreak (June 2024). [§7.1]
- Google DeepMind releases Gemma Scope: open sparse autoencoders on every layer of Gemma 2, making feature-level analysis reproducible outside frontier labs (August 2024).
- Bereska and Gavves publish the field-wide review of mechanistic interpretability for AI safety (April 2024). [§8.3]
- Cloud et al. introduce gradient routing: masking gradients during training to localize designated capabilities into designated subregions of the network, so they can later be removed or gated - the first of the training-time control methods (October 2024). [§7.3, §8.3; Part I §4.4]
- Jiang et al. give a theoretical account of *why* concepts are linear: the implicit bias of gradient descent plus the log-odds structure of the next-token objective provably yield linear encodings in simplified settings (March 2024). [§5.6; Part I §4.4]

2025 - Model biology

- Betley et al. report emergent misalignment: fine-tuning a model to write insecure code produces broadly misaligned behavior far outside the training domain (February 2025) - the flagship example of narrow training signals generalizing into broad persona shifts, and a cautionary tale for fine-tuning as a control lever. [§7.3, §8.3]
- Anthropic publishes attribution-graph analyses of a production model (*On the Biology of a Large Language Model*, March 2025), tracing multi-step internal computations (planning in poetry, multilingual circuits, unfaithful reasoning) in Claude 3.5 Haiku.
- Arcuschin et al. show chain-of-thought unfaithfulness arises naturally - implicit post-hoc rationalization without adversarial prompts (March 2025). [§8.1]
- Bogdan, Macar, Nanda, and Conmy publish thought anchors: sentence-level causal analysis of reasoning traces via resampling and attention suppression (June 2025). [§8.1]
- Casademunt et al. publish concept-ablation fine-tuning: projecting out unwanted activation directions during fine-tuning steers which solution the optimizer selects, largely

preventing emergent misalignment with no change to data or loss (July 2025). [§7.3; Part I §4.4]

- Chen et al. publish persona vectors: directions controlling character traits (sycophancy, harmfulness), used to screen fine-tuning data and for *preventative steering* - adding the trait direction during training so the model absorbs the pressure without encoding the trait (July 2025). [§7.3; Part I §4.4]
- The field's framing consolidates around *model biology* and applied interpretability (forensics of deployed incidents, probe-based monitoring, model diffing, eval-awareness characterization), treating complete reverse-engineering as a horizon rather than a prerequisite. [§8.3]

2026 - Control moves into the training loop

- Moskvoretskii et al. trace persona vectors through *pretraining* checkpoints, asking when in training a character trait becomes linearly represented - opening the door to shaping traits before they consolidate rather than steering them afterward. [Part I §4.4]
- Riché et al. introduce inoculation adapters: isolating unwanted generalization from fine-tuning into a removable adapter module, with fewer side effects than steering the whole network. [Part I §4.4]

Reading the arc

Four trends organize the seven years. First, the **unit of analysis** moved from neurons (2019) to directions and features (2022–2023) to behaviors of whole models (2025) - expanding, notably, in both directions at once: more microscopic tools and more organism-level questions. Second, the **standard of evidence** tightened from probe correlations (2019) to ablations (2020–2022) to patching methodology and causal scrubbing (2022–2023) to faithfulness auditing of the tools themselves (2023–2025). Third, the **purpose** shifted from scientific curiosity toward deployment: monitoring, forensics, and steering as operational controls. Fourth, the youngest trend, **control moved upstream**: from inference-time patches (2023–2024) into the training loop itself (gradient routing, concept-ablation fine-tuning, preventative steering, 2024–2026), so that interpretability instruments now help decide what a model learns, not only explain what it learned. The through-line from neuroscience runs the whole length: each era imported a method (receptive fields, lesions, population geometry, sparse coding) and eventually re-learned the corresponding lesson about its limits.

References

Papers marked with a filename in brackets are archived as PDFs in `new_book/references/`.

Chapter 5 - Representation Geometry

1. Ansuini, A., Laio, A., Macke, J.H., Zoccolan, D. (2019). *Intrinsic dimension of data representations in deep neural networks.* NeurIPS 2019. arXiv:1905.12784. [`ansuini2019_intrinsic_dimension.pdf`]

2. Park, K., Choe, Y.J., Veitch, V. (2023). *The Linear Representation Hypothesis and the Geometry of Large Language Models.* ICML 2024. arXiv:2311.03658. [`park2023_linear_representation_hypothesis.pdf`]

3. Elhage, N., Hume, T., Olsson, C., et al. (2022). *Toy Models of Superposition.* Transformer Circuits Thread / arXiv:2209.10652. [`elhage2022_toy_models_superposition.pdf`]

4. Facco, E., d'Errico, M., Rodriguez, A., Laio, A. (2017). *Estimating the intrinsic dimension of datasets by a minimal neighborhood information.* Scientific Reports 7. (The TwoNN estimator of §5.3.)

5. Jiang, Y., Rajendran, G., Ravikumar, P., Aragam, B., Veitch, V. (2024). *On the Origins of Linear Representations in Large Language Models.* ICML 2024. arXiv:2403.03867. [`jiang2024_origins_linear_representations.pdf`]

6. Tamkin, A., Taufeeque, M., Goodman, N.D. (2023). *Codebook Features: Sparse and Discrete Interpretability for Neural Networks.* ICML 2024. arXiv:2310.17230. [`tamkin2023_codebook_features.pdf`]

7. Hewitt, J., Thickstun, J., Manning, C.D., Liang, P. (2023). *Backpack Language Models.* ACL 2023. arXiv:2305.16765. [`hewitt2023_backpack_language_models.pdf`]

8. Jermyn, A.S., Schiefer, N., Hubinger, E. (2022). *Engineering Monosemanticity in Toy Models.* arXiv:2211.09169. [`jermyn2022_engineering_monosemanticity.pdf`]

9. Mikolov, T., Yih, W., Zweig, G. (2013). *Linguistic Regularities in Continuous Space Word Representations.* NAACL 2013.

10. Engels, J., Michaud, E.J., Liao, I., Gurnee, W., Tegmark, M. (2025). *Not All Language Model Features Are One-Dimensionally Linear.* ICLR 2025. arXiv:2405.14860.

11. Belrose, N., Furman, Z., Smith, L., et al. (2023). *Eliciting Latent Predictions from Transformers with the Tuned Lens.* arXiv:2303.08112.

12. Lad, V., Gurnee, W., Tegmark, M. (2024). *The Remarkable Robustness of LLMs: Stages of Inference?* arXiv:2406.19384.

13. Lenc, K., Vedaldi, A. (2015). *Understanding Image Representations by Measuring Their Equivariance and Equivalence.* CVPR 2015.

14. Bansal, Y., Nakkiran, P., Barak, B. (2021). *Revisiting Model Stitching to Compare Neural Representations.* NeurIPS 2021. arXiv:2106.07682.

15. Li, Y., Michaud, E.J., Baek, D.D., Engels, J., Sun, X., Tegmark, M. (2025). *The Geometry of Concepts: Sparse Autoencoder Feature Structure.* Entropy 27(3):344. arXiv:2410.19750.

16. Engels, J., Riggs, L., Tegmark, M. (2024). *Decomposing The Dark Matter of Sparse Autoencoders.* arXiv:2410.14670.

17. Chanin, D., et al. (2024). *A is for Absorption: Studying Feature Splitting and Absorption in Sparse Autoencoders.* arXiv:2409.14507.

18. Kantamneni, S., et al. (2025). *Are Sparse Autoencoders Useful? A Case Study in Sparse Probing.* ICML 2025. arXiv:2502.16681.

19. Karvonen, A., et al. (2025). *SAEBench: A Comprehensive Benchmark for Sparse Autoencoders.* ICML 2025. arXiv:2503.09532.

20. Ethayarajh, K. (2019). *How Contextual are Contextualized Word Representations? Comparing the Geometry of BERT, ELMo, and GPT-2 Embeddings.* EMNLP-IJCNLP 2019. arXiv:1909.00512.

21. Gao, J., He, D., Tan, X., Qin, T., Wang, L., Liu, T.-Y. (2019). *Representation Degeneration Problem in Training Natural Language Generation Models.* ICLR 2019. arXiv:1907.12009.

22. Timkey, W., van Schijndel, M. (2021). *All Bark and No Bite: Rogue Dimensions in Transformer Language Models Obscure Representational Quality.* EMNLP 2021. arXiv:2109.04404.

23. van der Maaten, L., Hinton, G. (2008). *Visualizing Data using t-SNE.* JMLR 9:2579–2605.

24. Wattenberg, M., Viégas, F., Johnson, I. (2016). *How to Use t-SNE Effectively.* Distill.

25. Kobak, D., Linderman, G.C. (2021). *Initialization is critical for preserving global data structure in both t-SNE and UMAP.* Nature Biotechnology 39:156–157.

26. Huh, M., Cheung, B., Wang, T., Isola, P. (2024). *The Platonic Representation Hypothesis.* ICML 2024. arXiv:2405.07987.

27. Moschella, L., Maiorca, V., Fumero, M., Norelli, A., Locatello, F., Rodolà, E. (2023). *Relative Representations Enable Zero-Shot Latent Space Communication.* ICLR 2023. arXiv:2209.15430.

28. Jha, R., Zhang, C., Shmatikov, V., Morris, J.X. (2025). *Harnessing the Universal Geometry of Embeddings.* NeurIPS 2025. arXiv:2505.12540.

29. Gurnee, W., Horsley, T., Guo, Z.C., et al. (2024). *Universal Neurons in GPT2 Language Models.* TMLR 2024. arXiv:2401.12181.

30. Davari, M., et al. (2023). *Reliability of CKA as a Similarity Measure in Deep Learning.* ICLR 2023. arXiv:2210.16156.

31. Kornblith, S., Norouzi, M., Lee, H., Hinton, G. (2019). *Similarity of Neural Network Representations Revisited.* ICML 2019. arXiv:1905.00414.

32. Allegra, M., Facco, E., Denti, F., Laio, A., Mira, A. (2020). *Data segmentation based on the local intrinsic dimension.* Scientific Reports 10. (The Hidalgo heterogeneous-manifold estimator of §5.3.)

33. Linzen, T. (2016). *Issues in evaluating semantic spaces using word analogies.* RepEval 2016.

34. Wendler, C., Veselovsky, V., Monea, G., West, R. (2024). *Do Llamas Work in English? On the Latent Language of Multilingual Transformers.* ACL 2024. arXiv:2402.10588.

35. Wu, Z., Yu, X.V., Yogatama, D., Lu, J., Kim, Y. (2025). *The Semantic Hub Hypothesis: Language Models Share Semantic Representations Across Languages and Modalities.* ICLR 2025. arXiv:2411.04986.

36. Chang, T.A., Tu, Z., Bergen, B.K. (2022). *The Geometry of Multilingual Language Model Representations.* EMNLP 2022. arXiv:2205.10964.

37. Libovický, J., Rosa, R., Fraser, A. (2020). *On the Language Neutrality of Pre-trained Multilingual Representations.* Findings of EMNLP 2020.

38. Tang, T., Luo, W., Huang, H., et al. (2024). *Language-Specific Neurons: The Key to Multilingual Capabilities in Large Language Models.* ACL 2024.

39. López, J.A.H., et al. (2022). *AST-Probe: Recovering Abstract Syntax Trees from Hidden Representations of Pre-trained Language Models.* ASE 2022. arXiv:2206.11719.

40. Feng, J., Steinhardt, J. (2024). *How do Language Models Bind Entities in Context?* ICLR 2024. arXiv:2310.17191.

41. Prakash, N., et al. (2024). *Fine-Tuning Enhances Existing Mechanisms: A Case Study on Entity Tracking.* ICLR 2024.

42. Liu, L., Liu, X., Gao, J., Chen, W., Han, J. (2020). *Understanding the Difficulty of Training Transformers.* EMNLP 2020. arXiv:2004.08249. (The amplification effect of §5.7.)

43. Zhu, D., Huang, H., Huang, Z., Zeng, Y., Mao, Y., Wu, B., Min, Q., Zhou, X. (2024). *Hyper-Connections.* arXiv:2409.19606. (The seesaw effect between gradient vanishing and representation collapse of §5.7.)

Chapter 6 - Features and Circuits

44. Cunningham, H., Ewart, A., Riggs, L., Huben, R., Sharkey, L. (2023). *Sparse Autoencoders Find Highly Interpretable Features in Language Models.* ICLR 2024. arXiv:2309.08600. [`cunningham2023_sparse_autoencoders.pdf`]

45. Gao, L., la Tour, T.D., Tillman, H., et al. (2024). *Scaling and Evaluating Sparse Autoencoders.* arXiv:2406.04093. [`gao2024_scaling_evaluating_saes.pdf`]

46. Olsson, C., Elhage, N., Nanda, N., et al. (2022). *In-context Learning and Induction Heads.* Transformer Circuits Thread / arXiv:2209.11895. [`olsson2022_induction_heads.pdf`]

47. Wang, K., Variengien, A., Conmy, A., Shlegeris, B., Steinhardt, J. (2022). *Interpretability in the Wild: a Circuit for Indirect Object Identification in GPT-2 small.* ICLR 2023. arXiv:2211.00593. [wang2022_ioi_circuit.pdf]

48. Zhang, F., Nanda, N. (2023). *Towards Best Practices of Activation Patching in Language Models: Metrics and Methods.* ICLR 2024. arXiv:2309.16042. [zhang2023_activation_patching_best_practices.pdf]

49. Conmy, A., Mavor-Parker, A.N., Lynch, A., Heimersheim, S., Garriga-Alonso, A. (2023). *Towards Automated Circuit Discovery for Mechanistic Interpretability.* NeurIPS 2023. arXiv:2304.14997.

50. Syed, A., Rager, C., Conmy, A. (2023). *Attribution Patching Outperforms Automated Circuit Discovery.* arXiv:2310.10348.

Chapter 7 - Control and Steering

51. Zou, A., Phan, L., Chen, S., et al. (2023). *Representation Engineering: A Top-Down Approach to AI Transparency.* arXiv:2310.01405. [zou2023_representation_engineering.pdf]

52. Turner, A.M., Thiergart, L., Leech, G., et al. (2023). *Activation Addition: Steering Language Models Without Optimization.* arXiv:2308.10248. [turner2023_activation_addition.pdf]

53. Arditi, A., Obeso, O., Syed, A., et al. (2024). *Refusal in Language Models Is Mediated by a Single Direction.* NeurIPS 2024. arXiv:2406.11717. [arditi2024_refusal_single_direction.pdf]

54. Jaeger, H. (2014). *Controlling Recurrent Neural Networks by Conceptors.* arXiv:1403.3369. [jaeger2014_conceptors.pdf]

55. Singh, S., Ravfogel, S., Herzig, J., et al. (2024). *Representation Surgery: Theory and Practice of Affine Steering.* ICML 2024. arXiv:2402.09631. [singh2024_representation_surgery_affine_steering.pdf]

56. Belrose, N., Schneider-Joseph, D., Ravfogel, S., Cotterell, R., Raff, E., Biderman, S. (2023). *LEACE: Perfect linear concept erasure in closed form.* NeurIPS 2023. arXiv:2306.03819.

57. Betley, J., Tan, D., Warncke, N., et al. (2025). *Emergent Misalignment: Narrow fine-tuning can produce broadly misaligned LLMs.* arXiv:2502.17424. [betley2025_emergent_misalignment.pdf]

58. Cloud, A., Goldman-Wetzler, J., Wybitul, E., Miller, J., Turner, A.M. (2024). *Gradient Routing: Masking Gradients to Localize Computation in Neural Networks.* arXiv:2410.04332. [cloud2024_gradient_routing.pdf]

59. Casademunt, H., Juang, C., Karvonen, A., Marks, S., Rajamanoharan, S., Nanda, N. (2025). *Steering Out-of-Distribution Generalization with Concept Ablation Fine-Tuning.* arXiv:2507.16795. [casademunt2025_concept_ablation_finetuning.pdf]

60. Chen, R., Arditi, A., Sleight, H., Evans, O., Lindsey, J. (2025). *Persona Vectors: Monitoring and Controlling Character Traits in Language Models.* arXiv:2507.21509. [chen2025_persona_vectors.pdf]

Chapter 8 - Evaluation and Frontiers

61. Lanham, T., Chen, A., Radhakrishnan, A., et al. (2023). *Measuring Faithfulness in Chain-of-Thought Reasoning.* arXiv:2307.13702. [lanham2023_measuring_cot_faithfulness.pdf]

62. Arcuschin, I., Janiak, J., Krzyzanowski, R., et al. (2025). *Chain-of-Thought Reasoning In The Wild Is Not Always Faithful.* arXiv:2503.08679. [arcuschin2025_cot_wild_not_faithful.pdf]

63. Bogdan, P.C., Macar, U., Nanda, N., Conmy, A. (2025). *Thought Anchors: Which LLM Reasoning Steps Matter?* arXiv:2506.19143. [bogdan2025_thought_anchors.pdf]

64. Bereska, L., Gavves, E. (2024). *Mechanistic Interpretability for AI Safety - A Review.* TMLR 2024. arXiv:2404.14082. [bereska2024_mech_interp_safety_review.pdf]

65. Power, A., Burda, Y., Edwards, H., Babuschkin, I., Misra, V. (2022). *Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets.* arXiv:2201.02177. [power2022_grokking.pdf]

66. Michaud, E.J., Liu, Z., Girit, U., Tegmark, M. (2023). *The Quantization Model of Neural Scaling.* NeurIPS 2023. arXiv:2303.13506. [michaud2023_quantization_neural_scaling.pdf]

67. Amari, S. (1998). *Natural Gradient Works Efficiently in Learning.* Neural Computation 10(2). (Treated in full in Part I, Chapter 4.)

68. Kirkpatrick, J., Pascanu, R., Rabinowitz, N., et al. (2017). *Overcoming catastrophic forgetting in neural networks.* PNAS 114(13).

69. Nanda, N., Chan, L., Lieberum, T., Smith, J., Steinhardt, J. (2023). *Progress measures for grokking via mechanistic interpretability.* ICLR 2023. arXiv:2301.05217. [nanda2023_grokking_progress_measures.pdf]

70. Moskvoretskii, V., Glandorf, D., Medina Moreira, J., Käser, T., West, R. (2026). *Tracing Persona Vectors Through LLM Pretraining.* arXiv:2605.13329. [moskvoretskii2026_tracing_persona_vectors.pdf]

71. Riché, M., Tan, D., Kohonen, V., Warncke, N., et al. (2026). *Inoculation Adapters: Improved Selective Generalization of Capabilities with Fewer Surprising Backdoors.* arXiv:2606.30252. [riche2026_inoculation_adapters.pdf]

Timeline sources (Appendix A)

72. Tenney, I., Das, D., Pavlick, E. (2019). *BERT Rediscovers the Classical NLP Pipeline.* ACL 2019. arXiv:1905.05950.

73. Hewitt, J., Liang, P. (2019). *Designing and Interpreting Probes with Control Tasks.* EMNLP 2019. arXiv:1909.03368.

74. Meng, K., Bau, D., Andonian, A., Belinkov, Y. (2022). *Locating and Editing Factual Associations in GPT.* NeurIPS 2022. arXiv:2202.05262.

75. Lieberum, T., Rajamanoharan, S., Conmy, A., et al. (2024). *Gemma Scope: Open Sparse Autoencoders Everywhere All At Once on Gemma 2.* arXiv:2408.05147.

Neuroscience

76. Kriegeskorte, N., Mur, M., Bandettini, P. (2008). *Representational similarity analysis - connecting the branches of systems neuroscience.* Frontiers in Systems Neuroscience 2:4. [`kriegeskorte2008_rsa.pdf`]

77. Jonas, E., Kording, K.P. (2017). *Could a Neuroscientist Understand a Microprocessor?* PLoS Computational Biology 13(1). [`jonas2017_neuroscientist_microprocessor.pdf`]

78. Barrett, D.G.T., Morcos, A.S., Macke, J.H. (2019). *Analyzing biological and artificial neural networks: challenges with opportunities for synergy?* Current Opinion in Neurobiology 55. arXiv:1810.13373. [`barrett2018_analyzing_biological_artificial_nns.pdf`]

79. Lindsay, G.W. (2021). *Convolutional Neural Networks as a Model of the Visual System: Past, Present, and Future.* Journal of Cognitive Neuroscience 33(10). arXiv:2001.07092. [`lindsay2020_cnns_model_visual_system.pdf`]

80. Sucholutsky, I., Muttenthaler, L., Weller, A., et al. (2023). *Getting aligned on representational alignment.* arXiv:2310.13018. [`sucholutsky2023_representational_alignment.pdf`]

81. Hubel, D.H., Wiesel, T.N. (1959). *Receptive fields of single neurones in the cat's striate cortex.* Journal of Physiology 148; and (1962) *Receptive fields, binocular interaction and functional architecture in the cat's visual cortex.* Journal of Physiology 160.

82. Salzman, C.D., Britten, K.H., Newsome, W.T. (1990). *Cortical microstimulation influences perceptual judgements of motion direction.* Nature 346.

83. Olshausen, B.A., Field, D.J. (1996). *Emergence of simple-cell receptive field properties by learning a sparse code for natural images.* Nature 381.

84. Kanwisher, N., McDermott, J., Chun, M.M. (1997). *The fusiform face area: a module in human extrastriate cortex specialized for face perception.* Journal of Neuroscience 17(11).

85. Churchland, M.M., Cunningham, J.P., Kaufman, M.T., et al. (2012). *Neural population dynamics during reaching.* Nature 487.

86. Rigotti, M., Barak, O., Warden, M.R., et al. (2013). *The importance of mixed selectivity in complex cognitive tasks.* Nature 497.

Web-only resources

87. Bricken, T., et al. (2023). *Towards Monosemanticity: Decomposing Language Models With Dictionary Learning.* `transformer-circuits.pub/2023/monosemantic-features`.

88. Templeton, A., et al. (2024). *Scaling Monosemanticity: Extracting Interpretable Features from Claude 3 Sonnet.* `transformer-circuits.pub/2024/scaling-monosemanticity`.

89. Chan, L., et al. (2022). *Causal Scrubbing: a method for rigorously testing interpretability hypotheses.* AI Alignment Forum.

90. Olah, C., et al. (2020). *Zoom In: An Introduction to Circuits.* Distill. `distill.pub/2020/circuits/zoom-in`.

91. Neuronpedia. Interactive SAE feature browser. `neuronpedia.org`.

92. nostalgebraist (2020). *interpreting GPT: the logit lens.* LessWrong, August 2020.

93. Elhage, N., Nanda, N., Olsson, C., et al. (2021). *A Mathematical Framework for Transformer Circuits.* Transformer Circuits Thread. `transformer-circuits.pub/2021/framework`.

94. Lindsey, J., Gurnee, W., Ameisen, E., et al. (2025). *On the Biology of a Large Language Model.* Transformer Circuits Thread. `transformer-circuits.pub/2025/attribution-graphs/biology`.