

000 SEMANTIC REGEXES: AUTO-INTERPRETING LLM 001 FEATURES WITH A STRUCTURED LANGUAGE 002

003 **Anonymous authors**
004

005 Paper under double-blind review
006

007 ABSTRACT 008

009 Automated interpretability aims to translate large language model (LLM) features
010 into human understandable descriptions. However, natural language feature de-
011 scriptions are often vague, inconsistent, and require manual relabeling. In re-
012 sponse, we introduce *semantic regexes*, structured language **descriptions of LLM**
013 features. By combining primitives that capture linguistic and semantic patterns
014 with modifiers for contextualization, composition, and quantification, semantic
015 **regexes** produce precise and expressive feature descriptions. Across quantitative
016 benchmarks and qualitative analyses, semantic regexes match the accuracy of nat-
017 ural language while yielding more concise and consistent feature descriptions.
018 Their inherent structure affords new types of analyses, including quantifying fea-
019 ture complexity across layers, scaling automated interpretability from insights into
020 individual features to model-wide patterns. Finally, in user studies, we find that
021 semantic regexes help people build accurate mental models of LLM features.
022

023 1 INTRODUCTION

024 AI models represent their learned concepts, like the “*Golden Gate Bridge*”, as linear directions in
025 latent space, called *features* (Bricken et al., 2023). Understanding a model’s features helps us an-
026 ticipate its behavior, assess its human-alignment, and intervene to ensure safe outcomes (Templeton
027 et al., 2024). Towards this goal, automated interpretability assigns human-readable descriptions
028 to each feature by analyzing patterns in its response to input data (Paulo et al., 2024; Bills et al.,
029 2023; Lin, 2023). With feature descriptions, researchers have identified how models encode domain-
030 relevant concepts, like protein structures (Gujral et al., 2025), and reconstructed circuits of features
031 that correspond to complex behaviors, like medical diagnoses (Lindsey et al., 2025).
032

033 Despite the advantages of feature descriptions, natural language is an imprecise interface for describ-
034 ing the computational roles features play in a model’s inference. Current automated interpretability
035 methods often yield overly verbose or inconsistent descriptions (Huang et al., 2023), reflecting the
036 difficulty of capturing tightly bounded feature behaviors in free-form text. Moreover, because nat-
037 ural language is prone to ambiguity, it is poorly suited for interpretability tasks that require composi-
038 tional reasoning, such as studying feature complexity or identifying redundant features. As a result,
039 even recent work identifying feature circuits in LLMs report needing skilled human relabeling to
040 describe a feature’s role in the network (Ameisen et al., 2025).
041

042 In contrast, structured languages offer well-defined syntax and semantics (Chomsky, 1956). By com-
043 bining a constrained set of primitives with compositional rules, structured languages (e.g., regular
044 language and programming languages) can capture precise patterns while maintaining expressiv-
045 ity (Lawson, 2005). Their grammatical structure provides additional affordances, including consis-
046 tent ways of expressing the same pattern (Knuth, 1965), the ability to concisely capture complex pat-
047 terns (Lawson, 2005), and mechanisms for representing abstraction (Backus, 1959; Aho & Ullman,
048 1972). As a result, structured languages can construct complex yet specific expressions, ranging
049 from regular expressions (Lawson, 2005) to software (Python Software Foundation, 2025).
050

051 To leverage the affordances of structured language in automated interpretability, we introduce *se-
052 mantic regexes*. The semantic regex language is designed to capture the diverse activation patterns
053 of LLM features, while providing the additional affordances of a structured language. Its primitives
054 are grounded in commonly observed feature functions, including exact token matches (`symbols`),
055 syntactic variants of words and phrases (`lexemes`), and broader semantic relationships (`fields`).
056

054 To enable greater expressivity, we extend these primitives with modifiers for contextual modification,
 055 composition, and quantification. As a result, the semantic regex language can express a range
 056 of features, from early-layer token detectors (e.g., “*on*” \rightarrow [:symbol on:]) to later-layer features
 057 that capture complex linguistic phenomena (e.g., “*the last name of a politician when it proceeds*
 058 *their title*” \rightarrow [:field political title:][:field last name:]).

059 Across automated interpretability evaluations, we find that semantic regexes as accurate as natural
 060 language, showing that constraining feature descriptions to the semantic regex language does
 061 not reduce expressivity. Beyond accuracy, their structured form offers additional affordances for
 062 interpretability, including producing more concise feature descriptions and enforcing description
 063 consistency across functionally similar features. Moreover, since semantic regex primitives and
 064 modifiers exist across levels of abstraction, they serve as a proxy for feature complexity, allowing us
 065 to perform model-wise analysis of feature behavior. Finally, in a user study, we find that semantic
 066 regexes help people build accurate mental models of LLM features.

068 2 RELATED WORK

070 Interpretability aims to expose LLMs’ internal concepts in order to explain their behavior and
 071 evaluate their human-alignment (Lipton, 2018; Bricken et al., 2023; Olah et al., 2020; 2017).
 072 LLMs are known to encode concepts along linear directions in latent space, often referred to as
 073 features (or latents) (Park et al., 2024; Elhage et al., 2022). Methods like sparse autoencoders
 074 (SAEs) (Huben et al., 2024; Gao et al., 2025; Rajamanoharan et al., 2024; Bussmann et al., 2025)
 075 and transcoders (Ameisen et al., 2025; Dunefsky et al., 2024) extract LLM features, and have un-
 076 covered human-interpretable features, including those that correspond to domain-specific and safety
 077 relevant concepts (Lindsey et al., 2025; Gujral et al., 2025; Bricken et al., 2023).

078 While features correspond to concepts, *which* concept a feature encodes is not obvious a priori.
 079 Early interpretability work approached this problem by manually analyzing the inputs that activate
 080 each feature (Olah et al., 2020; 2017; 2018; Carter et al., 2019). To scale this process, automated
 081 interpretability aims to automatically describe each feature’s concept (Bau et al., 2017; Hernandez
 082 et al., 2022; Bills et al., 2023; Shaham et al., 2024). Given a feature’s activating data, automated
 083 interpretability prompts a language model describe the feature (Bills et al., 2023; Paulo et al., 2024;
 084 Gur-Arieh et al., 2025; Lin, 2023). Aligned with this goal, our method adopts a similar pipeline for
 085 generating feature descriptions, but generates descriptions using a structured language.

086 Our approach is inspired by recent research showing that LLM features represent structured con-
 087 cepts. Activation analyses show that feature complexity increases with depth (Jin et al., 2025) and
 088 that models represent concepts across levels of abstraction (Boggust et al., 2025; Chanin et al., 2024;
 089 Bussmann et al., 2025; Zaigrajew et al., 2025). In response, researchers have proposed lightweight
 090 taxonomies that classify features by function (Ameisen et al., 2025; Lindsey et al., 2025; Gur-Arieh
 091 et al., 2025), and called for feature description formalisms (Huang et al., 2023). These findings mo-
 092 tivate our use of a structured language, enabling semantic regexes to describe a feature’s activation
 093 pattern and express its level of abstraction.

094 3 SEMANTIC REGEXES

095 Semantic regexes use a structured language to describe LLM features. Built around a system of
 096 human-interpretable primitives (Section 3.1.1) and modifiers (Section 3.1.2), semantic regexes cap-
 097 ture the low-level syntactic patterns and higher-level semantic concepts that LLM features represent.
 098 Unlike natural language, which is flexible but ambiguous, the semantic regex language restricts ex-
 099 pressivity to ensure the resulting feature descriptions explicitly convey their meaning. On the other
 100 hand, while inspired by regular expressions, the semantic regex language is not a regular language
 101 and extends beyond one-to-one character patterns to capture more abstract concepts.

104 3.1 THE SEMANTIC REGEX LANGUAGE

105 The semantic regex language consists of compositional components: *primitives* define the textual
 106 units a semantic regex matches, and *modifiers* refine or expand their scope (Figure 1). Together these
 107 components form a compact, yet expressive language for specifying LLM feature patterns.

108 109 110 111 112 113 114 115	Primitives symbol <i>Match exact strings</i> SEMANTIC REGEX: <code>[:symbol color:]</code> DATA EXAMPLES: a splash of color color your world	lexeme <i>Match syntactically related variants</i> SEMANTIC REGEX: <code>[:lexeme color:]</code> DATA EXAMPLES: color in a coloring book her favorite colors	field <i>Match semantically related variants</i> SEMANTIC REGEX: <code>[:field color:]</code> DATA EXAMPLES: blue jeans pink skies at night
116 117 118 119 120 121 122 123	Modifiers context <i>Match when appearing in the context</i> SEMANTIC REGEX: <code>@{:context politics:}[:field color:]</code> DATA EXAMPLES: turned blue in the election Green Party candidate	combinations <i>Sequential or OR combinations</i> SEMANTIC REGEX: <code>[:field color:] [:symbol and:] [:symbol or:] [:field color:]</code> DATA EXAMPLES: green or yellow bananas it is black and white	quantifiers <i>Metacharacter to denote zero or one</i> SEMANTIC REGEX: <code>[:symbol a:] [:field color:]?</code> <code>[:field flower:]</code> DATA EXAMPLES: a red rose is blooming it is a da isy

Figure 1: The semantic regex language consists of a set of primitives (top) that can be applied independently or combined with modifiers (bottom) to express diverse feature activation patterns.

We developed the semantic regex language using a grounded-theory approach (Corbin & Strauss, 1998), deriving its components from empirical analysis of real LLM features. By manually surveying of thousands of features across models, layers, and feature sources on Neuronpedia (Lin, 2023), we identified recurring patterns (e.g., context dependent activations). We introduced new primitives or modifiers only when they captured a recurring pattern, increased the language’s descriptive coverage, and preserved intelligibility of the language. We continued this iterative process until reaching saturation, resulting in a language capable of describing all the features we examined.

3.1.1 PRIMITIVES

Primitives are the atomic components of a semantic [regex](#). They specify the type of textual pattern a semantic regex will match, ranging from exact characters to categorical relationships.

Symbols Symbols (`[:symbol X:]`) match exact strings X . For instance, `[:symbol color:]` matches the string *color*, such as in **a** splash of **color**. These are the most specific and simplest primitives, and they commonly describe features that activate on specific tokens or phrases.

Lexemes Lexemes (`[:lexeme X:]`) match syntactic variants of X . [Drawing on](#) linguistics, a lexeme is the *abstract* form of a word [that encompasses](#) all of its surface variants, like changes in tense or plurality. For example, `[:lexeme color:]` matches *color*, *colors*, *coloring*, etc., such as **color** in a **coloring** book. Lexemes typically describe features that capture a word’s meaning.

Fields Fields (`[:field X:]`) match semantic variants of X . [Drawing on](#) linguistics, fields refer to words or phrases in a conceptual domain. For instance, `[:field color:]` matches *red*, *orange*, *blue*, etc., as in **blue** jeans. Fields often [apply to](#) features that [activate on a conceptual](#) category.

3.1.2 MODIFIERS

Modifiers refine and extend primitives with context, composition, and quantification. This increases the expressive power of semantic regexes, allowing them to represent a wider range of features.

Context Contexts (`@{:context X:} (semantic regex)`) match a semantic regex in the context X . For instance, `@{:context politics:}[:symbol color:]` only matches `[:symbol color:]` in a political context, matching **Green** Party but not **green** apple. Contexts help match features that represent domain-dependent concepts.

162 **Composition** Semantic regexes can compose in *sequence* or *alternation* (`|`). For in-
 163 stance, `[:field color:](:symbol and:)|[:symbol or:]` matches
 164 both `green` or `yellow` and `black` and `white`. Composing semantic regexes allows them to
 165 match more complex features while maintaining precision.
 166

167 **Quantification** Semantic [regexes](#) also make use of the regular expression quantifier zero or one (`?`).
 168 As an example, `[:symbol a:][:field color:]?[:field flower:]` matches `a` `red` `rose`
 169 and `a` `da` `isy`. Quantifiers allow additional flexibility in the semantic regex.
 170
 171

172 4 METHODS

173 To study how the semantic regex language alters feature interpretability, we embed it within a stan-
 174 dard automated interpretability pipeline (Paulo et al., 2024; Templeton et al., 2024; Puri et al., 2025;
 175 Bills et al., 2023). The pipeline consists of three components: a *subject* model whose features
 176 we describe, an *explainer* model that generates feature descriptions, and an *evaluator* model that
 177 scores them. Given a subject model feature and its activating data, we prompt an explainer [model](#) to
 178 produce a description in natural or semantic regex language and use the evaluator [model](#) to score
 179 how well the description matches the feature’s behavior. [Following prior interpretability work](#) (Bills
 180 et al., 2023; Gur-Arieh et al., 2025) and [our ablation study in Appendix B](#), we use [GPT-4o-mini](#) as
 181 the explainer and evaluator. This [pipeline](#) allows us to directly compare semantic regexes to natural
 182 language descriptions across varied axes of interpretability. [It also demonstrates that, by decoupling](#)
 183 [the format of the feature description from the generation process](#), semantic regexes are compatible
 184 with existing and future automated interpretability pipelines.
 185

186 4.1 COLLECTING MODEL FEATURES AND ACTIVATIONS

187 We study semantic regexes on two families of *subject* models, GPT-2 (Radford et al., 2019) and
 188 Gemma-2 (Mesnard et al., 2024). While our approach could describe model neurons (or any com-
 189 ponent matched to activating data), we focus on describing latent features identified by sparse coding
 190 methods (e.g., SAEs), since they often represent monosemantic concepts that are easier to describe
 191 and interpret (Bricken et al., 2023).
 192

193 **GPT-2** We apply semantic regexes to GPT-2-Small (Radford et al., 2019) and its residual layer
 194 features from Bloom (2024), identified using SAEs with 24,576 features (GPT-2-RES-25k).
 195

196 **Gemma** We also apply semantic regexes to Gemma-2-2B (Mesnard et al., 2024) and its
 197 Gemma Scope (Lieberum et al., 2024) residual layer features (Gemma-2-2B-RES-16k and
 198 Gemma-2-2B-RES-65k).
 199

200 To collect activating data for each feature, we use the Neuronpedia (Lin, 2023) [API](#). It provides
 201 activating data from OpenWebText (Gokaslan et al., 2019) for GPT-2-Small and Pile Uncopy-
 202 righted (Gao et al., 2021) for Gemma-2-2B.
 203

204 4.2 GENERATING FEATURE DESCRIPTIONS

205 Given features and their activating data, we prompt an *explainer* model ([GPT-4o-mini](#)) to generate
 206 feature descriptions. We compare our [semantic-regex](#) method against natural language feature
 207 description methods: `token-act-pair` (Bills et al., 2023) and `max-acts` (Paulo et al., 2024).
 208 These methods are commonly used and vary in how they generate descriptions, allowing us to get
 209 a comprehensive understanding of how semantic regexes compare to natural language descriptions.
 210 See Appendix F.1 for implementation details.
 211

212 **token-act-pair** is based on Bills et al. (2023) and its Neuronpedia implementation (Lin, 2023)
 213 (`oai_token-act-pair`). Here, the explainer model is prompted with the feature description
 214 task and three few-shot examples. Then, it is shown a feature’s top five activating examples and
 215 asked to continue the sentence “*the main thing this neuron does is find*”. Each example is displayed
 as a list of tokens and their normalized activation values, like “*these 8\n tokens 10\n activate 0*”.

216 **max-acts** follows Neuronpedia’s (Lin, 2023) implementation of Paulo et al. (2024)
 217 (`eleuther_acts_top2`). Instead of showing each token’s activation value, like in
 218 `token-act-pair`, it maintains a more natural text format by delimiting activating tokens within
 219 the text, like “`⟨⟨these tokens⟩⟩ activate`”. It prompts the explainer model with the feature descrip-
 220 tion task and three few-shot examples. Then, given the feature’s top 20 activating examples, the
 221 explainer model is asked to “*describe the text latents that are common in the examples*”.

222 **semantic-regex** aims to accurately describe LLM features and only differs from natural lan-
 223 guage methods in the language available to the explainer model. As a result, we simply inject
 224 `semantic-regex` specific instructions into existing prompting strategies. To take advantage of its
 225 efficient example formatting, we adapt `max-acts`’ prompt by updating the instructions to follow
 226 the semantic regex language, adding a concise definition of the semantic regex language, and chang-
 227 ing the few-shot examples to demonstrate semantic regex primitives and modifiers. We show the
 228 explainer model a feature’s top 10 activating examples and ask it to “*output a short explanation*
 229 *followed by a semantic regex*”, which we find improves the model’s ability to follow the syntax.

230
 231 **4.3 EVALUATING FEATURE DESCRIPTIONS**
 232

233 We evaluate feature descriptions using `common` automated interpretability metrics (Paulo et al.,
 234 2024; Puri et al., 2025). Each metric `uses` an *evaluator* model (`GPT-4o-mini`) to evaluate the
 235 descriptions’ `fidelity` to the feature’s behavior. *Generation* metrics test the description’s ability to
 236 generate activating examples (akin to precision), *discrimination* metrics test the description’s ability
 237 to match known activating examples (akin to recall), and *faithfulness* metrics test the description’s
 238 ability to match steered `generation` (a measure of causality). Implementation details in Appendix F.2.

239 **Generation metrics** test a feature description’s precision by evaluating its ability to generate acti-
 240 vating examples. We use `clarity` (Puri et al., 2025), which compares generated and random
 241 examples’ activations using the Gini index (a rescaling of the ROC AUC). Under this metric, overly
 242 broad descriptions score low by generating data outside the feature’s activation space, while ideal or
 243 overly narrow descriptions score high by only generating activating data.

244
 245 **Discrimination metrics** test the description’s recall by evaluating its ability to match known
 246 activating examples. We compute `detection` and `fuzzing` from Paulo et al. (2024) and
 247 `responsiveness` and `purity` from Puri et al. (2025). These metrics ask the evaluator model
 248 whether the feature description matches activating and random examples. Given the match results,
 249 `detection` measures balanced accuracy, `responsiveness` the Gini index, and `purity` average
 250 precision. Instead of matching the entire example, `fuzzing` asks if the description matches the ex-
 251 ample’s activating tokens and computes the balanced accuracy of these more specific match results.
 252 Under these metrics, overly narrow descriptions score low by missing activating examples, while
 253 ideal or overly broad `descriptions` score high by covering the entire activation space.

254 **Faithfulness metrics** test the description’s faithfulness to causal interventions on the feature. We
 255 use `faithfulness` (Puri et al., 2025), which asks the evaluation model whether the feature descrip-
 256 tion matches continuations of random text when the feature is steered versus ablated.

257
 258 **5 RESULTS**
 259

260 **5.1 SEMANTIC REGEXES ARE AS ACCURATE AS NATURAL LANGUAGE DESCRIPTIONS**
 261

262 The goal of automated interpretability is to generate feature descriptions that accurately characterize
 263 a feature’s activations. We benchmark semantic regexes against common natural language descrip-
 264 tion methods (`token-act-pair` and `max-acts`) using discrimination, generation, and faithfulness
 265 metrics. To ensure our results generalize across models and features, we evaluate 100 features per
 266 layer from GPT-2-RES-25k, Gemma-2-2B-RES-16k, and Gemma-2-2B-RES-65k.

267 We find that `semantic-regex` descriptions perform on par with natural language (Figure 2).
 268 Specifically, semantic regexes are non-inferior ($p < 0.05^1$) to natural language on `clarity` across

269 ¹ One-sided paired *t*-test with non-inferiority margin $\Delta = 5\%$ and Bonferroni correction for superiority.

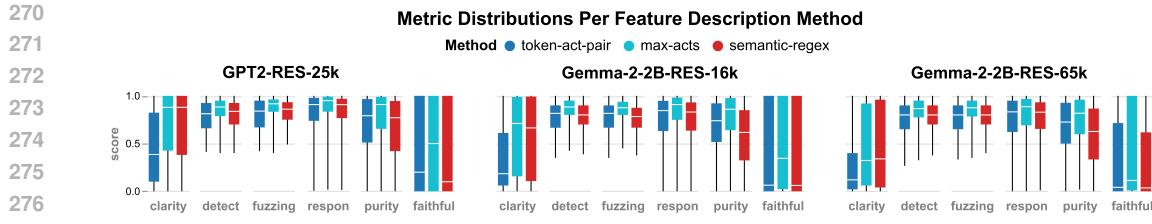


Figure 2: Semantic regexes perform on par with natural language feature descriptions across evaluations on GPT-2-RES-25k, Gemma-2-2B-RES-16k, and Gemma-2-2B-RES-65k, suggesting that the semantic regex language is appropriately expressive to describe LLM features.

all models, detection for GPT-2-RES-25k, and responsiveness for Gemma-2-2B-RES-65k. Moreover, semantic-regex outperforms token-act-pair ($p < 0.05^1$) on clarity across all models, and detection, fuzzing, and responsiveness on GPT-2-RES-25k and Gemma-2-2B-RES-65k. This is non-obvious, as the semantic regex language is significantly constrained compared to the hundreds of thousands of words available in natural language. Moreover, while the explanation and evaluation models are well-versed in natural language, they learned the semantic regex language via only a brief description and few-shot examples. These results suggest that imposing structure on feature descriptions does not reduce their accuracy, while offering advantages that we explore in the following sections.

5.2 SEMANTIC REGEXES IMPROVE CONCISENESS AND CONSISTENCY

While semantic regexes are similarly accurate to natural language descriptions, their structure offers distinct benefits for interpreting LLM feature behavior (Figure 3).

Semantic regexes are more concise than natural language descriptions. Concise feature descriptions adhere to explanation norms (Tim Miller, 2019), making them easier to scan and interpret. While natural language descriptions often require verbose phrases to capture a feature’s activation pattern (Huang et al., 2023), semantic regexes can encode the same information in more compact form using its signal-rich components. For example, in Figure 3 top, the verbose natural language description “*The presence of the sequence 54 indicating a year, time, or numeric reference frequently associated with events*” can be expressed as [:symbol 54:]. Quantitatively, semantic regexes are consistently shorter. Across 100 randomly sampled features per layer in GPT-2-RES-25k, Gemma-2-2B-RES-16k, and Gemma-2-2B-RES-65k, the median description length is 41 characters (IQR: 19–59) compared to 139 (IQR: 119–166) for max-acts and 55 (IQR: 46–66) for token-act-pair. This conciseness directly benefits interpretability, allowing human evaluators in Section 5.4 to parse shorter descriptions without distracting details.

Semantic regexes are more consistent than natural language descriptions. LLMs often contain redundant features that activate on similar inputs and serve the same function. In circuit identification applications, recognizing redundant features helps reduce circuit complexity and identify the complete mechanistic circuit (Ameisen et al., 2025). Since the semantic regex language constrains the space of allowable expressions, it produces more consistent descriptions for similar features, making redundancy easier to detect. For example, in Figure 3 middle, two redundant features from different layers of Gemma-2-2B-RES-16k both activate on the token “Advertisement”. While their natural language descriptions differ (“*the word Advertisement*” vs. “*advertisement markers*”), their semantic regex descriptions are identical: [:symbol Advertisement:].

To quantify this effect, we measure consistency by asking how often a method produces the same description when given different random samples of a feature’s activating data. This simulates redundancy, since the underlying feature is fixed but observed activating examples vary. Evaluating semantic-regex, max-acts, and token-act-pair on five random features per layer of GPT-2-RES-25k, each with five generated descriptions, we find that semantic-regex yields identical descriptions 33.6% of the time, compared to 12.2% for token-act-pair and 0.0% for max-acts. These results suggest that constraining the description space with semantic regexes improves consistency, making it easier to detect redundant features.



Figure 3: Semantic regexes are often more concise (top), more consistently describe equivalent features (middle), and better reflect feature complexity (bottom) than natural language descriptions.

5.3 SEMANTIC REGEXES REFLECT FEATURE COMPLEXITY

Beyond matching the accuracy of natural language descriptions and offering benefits like conciseness and consistency, the **structured format** of semantic regexes offers **additional** affordances for model interpretability. Each semantic regex is built from primitives that span increasing levels of abstraction, where symbols match specific characters, lexemes extend to syntactic variants, and fields encode semantic relationships. Moreover, the number of components in a semantic regex is also a proxy for feature complexity, where features described using a single primitive (e.g., [:symbol left:]) are typically conceptually simpler than features that require multiple compositions of primitives and modifiers (e.g., @{:context political affiliations:}[:symbol left:]|[:symbol right:]).

We use the level of abstraction encoded in semantic regexes to measure feature complexity, finding that features become more complex deeper in the model (Figure 4). We generate semantic regexes for 1,000 features per layer in GPT-2-RES-25k, Gemma-2-2B-RES-16k, and Gemma-2-2B-RES-65k. While early-layer features are described by smaller and simpler semantic regexes, longer and more abstract semantic regexes are needed to describe later-layer features. In particular, we find that the average number of components per semantic regex (i.e., symbols, lexemes, fields, and contexts) **steadily increases across layers**. This shift is mirrored in the **composition of a semantic regex**, where early layers have a greater proportion of single-primitive descriptions which decreases in favor of combinations of primitives and modifiers, particularly sequence and alternation compositions. Similarly, we observe that the types of primitives reflect increasing feature complexity. While all semantic regexes **contain** primitives, we see a decrease in low-level primitives.

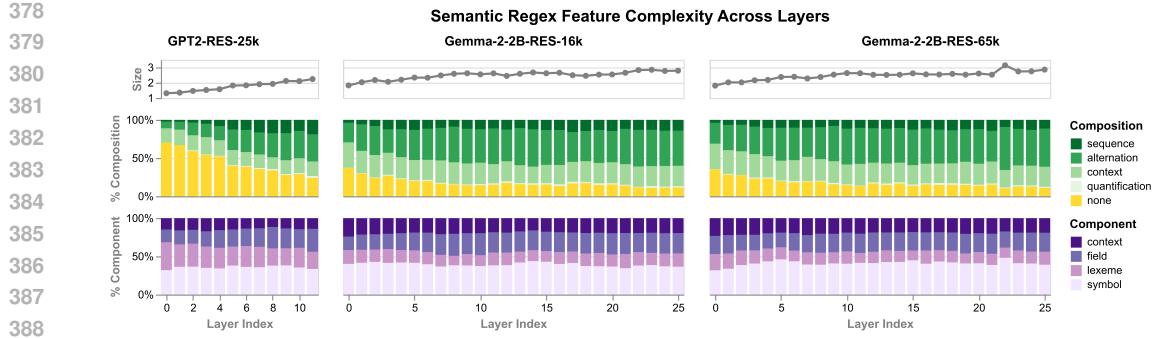


Figure 4: Semantic regexes encode feature complexity. The number (top) and abstraction (middle, bottom) of components increase across model layers, indicating increasingly complex features.

tives, like lexemes, and an increase in fields (the most abstract primitive). We show an example of increasing feature complexity from early to later layers in Figure 3 **bottom**.

Together, these trends indicate that later layers require [longer](#) and more abstract semantic regexes to capture their complex feature behaviors. This aligns with prior research demonstrating that later layers encode increasingly complex representations (Jin et al., 2025; Tenney et al., 2019; Sun et al., 2025). However, unlike prior methods that rely on model probes or feature testing, we are able to read this complexity directly from the semantic regex feature description. As a result, while like natural language descriptions, semantic regexes allow us understand individual features, their structure also allows us to interpret entire model attributes.

5.4 SEMANTIC REGEXES HELP PEOPLE BUILD MENTAL MODELS OF LLM FEATURES

A common role of feature descriptions is to convey the feature’s behavior to a human interpreter (Ameisen et al., 2025; Lin, 2023). Thus, we investigate how semantic regexes impact people’s mental models of LLM features (full protocol in Appendix E). We conducted a 24-person study with AI experts who were representative of people most likely to use feature descriptions in practice. To obtain insights across a range of features, we used 12 GPT-2-RES-25k features that had diverse activation patterns and accurate natural language and semantic regex descriptions. Given a description, participants were asked to generate three activating phrases and one near-miss counterfactual. Each participant generated phrases for three max-acts and three semantic-regex feature descriptions, resulting in 425 positive and 143 counterfactual phrases.

Forming an accurate mental model of an LLM feature means being able to express the feature’s decision boundary, i.e., what does and does not activate the feature. We quantify this by measuring the difference between the feature’s maximum activation on each participant’s positive **phrases** and their counterfactual **phrase**. Small or negative values indicate that the participant had difficulty distinguishing activating and non-activating phrases, while large positive values reflect an understanding of the feature’s decision boundary. We compare the mean **differences of max-acts** and **semantic-regex** descriptions of the same feature, **finding that** participants scored higher using semantic regex descriptions on 9 of 12 features (Figure 5).

Although both description types were accurate, natural language often introduced extraneous details that misled participants. For instance, one feature activated on variants of the phrase “expected to”, but its natural language description included “‘expected to’ is frequently used to indicate anticipation or prediction regarding future events or outcomes”. A participant over-indexed on the additional detail, expecting the highly activating phrase “*He does not know meaning of the phrase expected to*” to be a counterexample because it did not indicate anticipation. In contrast, [:lexeme expect :][:symbol to:] more concisely expresses the activation pattern (see Section 5.2) and enabled participants to generate strongly activating positives and non-activating counterfactuals.

We also find that semantic regexes reduce ambiguity by conveying activation patterns via example. For a feature that activates on times followed by “p.m.”, the semantic regex specifies this pattern directly as [:field time:] [:symbol p.m. :], and as a result, every participant included “p.m.”

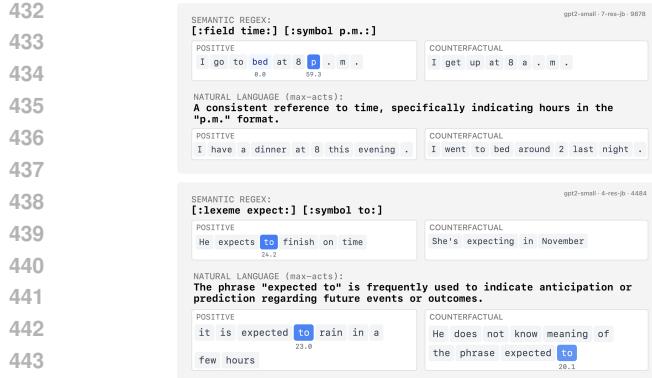


Figure 5: With semantic regexes, user study participants generated strongly activating positive examples and non-activating counterfactuals, indicating their understanding of the feature.

in their positive phrases. In contrast, natural language descriptions can leave room for interpretation, where one participant misinterpreted “*reference to time, specifically indicating hours in the ‘p.m.’ format*” as any time in the evening and left off the critical “*p.m.*”. A similar pattern occurred in a feature for political conjunctions, where the semantic regex directly included [:symbol and:] | [:symbol or:], making the intended conjunction pattern unambiguous.

While semantic regexes’ concise examples generally benefited participants, there were cases where the verbosity of natural language was more effective. In the two features where natural language substantially outperformed semantic regexes, the semantic regexes were accurate but too minimal to elicit strongly activating phrases from participants. For example, one feature activates on days of the week used within full sentences. The conciseness of the semantic regex [:field days of the week:] led many participants to produce single-word outputs, like “*Monday*”, which activated **slightly**. However, the natural language description “*days of the week indicating specific events*” prompted longer phrases **with larger activations**.

Finally, we find that participants were able to understand and use semantic regexes with minimal instruction. Despite receiving only a short description and a single example, participants generally interpreted the language correctly. In fact, we received more clarification questions about how to interpret the natural language descriptions than semantic regexes. This contrasts prior work suggesting that structured languages come at a cost because they require “specialized training” (Huang et al., 2023), and instead signals their promise as tools for LLM interpretability.

6 DISCUSSION AND LIMITATIONS

Semantic regexes provide a structured syntax for describing LLM features. In doing so, they result in feature descriptions that are more consistent and concise than natural language while still matching its expressive power. However, designing this structured language introduces trade-offs that create both benefits and limitations of semantic regexes and point to several directions for future work.

Since natural language descriptions often contain irrelevant details, we designed semantic regexes to be concise. While this generally helps identify the pertinent activation pattern, it can produce overly terse descriptions. For example, [:field musicians:] describes a feature that activates on famous musicians like “*Taylor Swift*”, incorrectly implying that “*guitarist*” would strongly activate the feature. Striking a balance between conciseness and expressivity across all features may involve adjusting model prompts, extending the language to encode hierarchical concepts (e.g., [:field musician.name:]), or using validation loops Shaham et al. (2024) to mitigate ambiguity.

Additionally, although semantic regexes increase consistency, they do not enforce a unique mapping from activation pattern to description. Many valid semantic regexes can describe the same feature, even if some are less readable, like @{:context Germany:}[:symbol German:]). This non-uniqueness is not inherently problematic. In programming languages, for instance, there are many equivalent ways to implement the same function. However, languages often develop style guides

486 that suggest the most readable syntax (van Rossum et al., 2001). Similar heuristics may benefit the
 487 semantic regex language, particularly when people (rather than models) are the intended interpreters.
 488

489 To keep the semantic regex syntax minimal, we leave some components underspecified. For ex-
 490 ample, `symbol` match exact strings, but the semantic regex language does not specify if they also
 491 match string variations (e.g., “*fruit*” and “*Fruit*”). This simplicity avoids an overly complex vocab-
 492 ular but can cause the model to make inconsistent assumptions across features (e.g., sometimes
 493 outputting `[:symbol fruit:]` and other times `[:symbol fruit:]||[:symbol Fruit:]`). In-
 494 corporating additional components, like case-insensitive flags, could reduce this ambiguity, espe-
 495 cially as growing familiarity with semantic regexes may allow for more complex syntax.
 496

497 These limitations also impact feature description evaluations. Since current metrics rely on discrete
 498 judgments of the feature description, issues like non-uniqueness and ambiguity can lead to false neg-
 499 atives and false positives for both semantic regexes and natural language methods. Developing more
 500 expressive metrics, such as continuous scoring schemes or readability evaluations, could provide a
 501 more comprehensive understanding of the differences between feature description methods.
 502

503 Although our results indicate that semantic regexes are expressive, structured, and helpful for human
 504 interpretation, additional evaluations could deepen our understanding of feature descriptions. While
 505 our repeated evaluations show that semantic regexes produce consistent descriptions, large-scale
 506 repetitions of the quantitative analysis could measure pipeline stability. Additionally, our user study
 507 provides qualitative insights, but larger crowdsourced studies could statistically quantify the value
 508 of semantic regexes and whether LLM-as-a-judge evaluations faithfully capture human preferences
 509 in this domain. These findings could inform improvements to future feature description pipelines.
 510

511 Moreover, semantic regexes are not a solution to polysemy. While the OR modifier helps
 512 capture simple polysemic features (e.g., `[:field clothing:]||[:field Ernest:]`), the
 513 current pipeline struggles with high degrees of concept entanglement, often resulting in inco-
 514 herent descriptions, like `@{[:context syntax:]}{[:symbol by:]||[:symbol (:||[:symbol
 515 .:]||[:symbol last:]||[:symbol then:]||[:symbol Count:]||[:symbol ->:]}`. How-
 516 ever, since semantic regexes are agnostic to the generation pipeline, they could easily slot into newly
 517 developed methods for disentangling polysemic activations (Kopf et al., 2025).
 518

519 Finally, semantic regexes require models to learn the semantic regex language from only a brief
 520 description and a few examples. As a result, we observe “grammatical” errors in generation (e.g.,
 521 applying the wrong primitive) and evaluation (e.g., misinterpreting a primitive and making an incor-
 522 rect match). Although models also make mistakes on natural language descriptions, they typically
 523 stem from misidentifying the activation pattern rather than misunderstanding the language. This gap
 524 raises important questions for automated interpretability. As we increasingly think of LLM features
 525 as computational units, structured language should better capture their patterns (akin to how pro-
 526 grams better represent algorithms). However, if automated pipelines rely on models biased towards
 527 natural language, structured languages face a substantial barrier to adoption. Moreover, people eas-
 528 ily learn structured languages, so it is unclear whether automated evaluations reflect human ability.
 529 Future work could investigate these discrepancies to understand how model and human reasoning
 530 diverge on novel structured languages and its impact on interpretability pipelines.
 531

532 7 CONCLUSION

533 We introduce semantic `regexes`, a structured language for automatically describing LLM features.
 534 The semantic regex language is grounded in current understanding of LLM features. Each primitive
 535 and modifier is designed to reflect patterns observed in interpretability research—i.e., that
 536 features often respond to exact tokens (`symbol`), syntactic word forms (`lexeme`), and semantic cat-
 537 egories (`field`), and their activations are `domain-dependent` (`context`) and `co-occur` with other
 538 patterns (Lin, 2023; Templeton et al., 2024). As our understanding of LLM representations grows,
 539 we expect the semantic regex language will evolve, with components added or altered to capture
 540 new feature behaviors. Just as there are many `programming` languages `to meet different goals`, future
 541 languages for interpretability may be developed with affordances `suites to particular` interpretability
 542 `tasks`, such as highlighting input vs. output features, describing particular model components (e.g.,
 543 attention heads), or exposing safety-relevant features. The affordances of consistency, conciseness,
 544 and `complexity` that we build into semantic regexes expose a broader design space of structured
 545 languages for interpretability that `can` improve our collective understanding and control of LLMs.
 546

540 ETHICS STATEMENT

541

542 As part of this research, we conducted a user study where we surveyed 24 participants from within
 543 our institution (Section 5.4). All participants gave informed consent and were informed they could
 544 withdraw at any time. We did not collect any identifying information and all participant responses
 545 were anonymized. This study was approved under our institution’s internal user survey policies.

546

547 REPRODUCIBILITY STATEMENT

548

549 Implementation details for our method and the baseline methods are in Appendix F.1, including the
 550 full prompts and hyperparameters. Implementation details for the evaluation metrics are provided
 551 in Appendix F.2, including full prompts and hyperparameters. Code is available at `redacted`, and
 552 an interactive interface displaying our results is available at `redacted`. Details to recreate our user
 553 study, including our survey instructions, are listed in Appendix E.

554

555 REFERENCES

556

557 Alfred V. Aho and Jeffrey D. Ullman. *The theory of parsing, translation, and compiling. 1: Pars-
 558 ing*. Prentice-Hall, 1972. ISBN 0139145567. URL <https://www.worldcat.org/oclc/310805937>.

559

560 Emmanuel Ameisen, Jack Lindsey, Adam Pearce, Wes Gurnee, Nicholas L. Turner, Brian Chen,
 561 Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, Jonathan Marcus, Michael Sklar,
 562 Adly Templeton, Trenton Bricken, Callum McDougall, Hoagy Cunningham, Thomas Henighan,
 563 Adam Jermyn, Andy Jones, Andrew Persic, Zhenyi Qi, T. Ben Thompson, Sam Zimmerman,
 564 Kelley Rivoire, Thomas Conerly, Chris Olah, and Joshua Batson. Circuit tracing: Revealing
 565 computational graphs in language models. *Transformer Circuits Thread*, 2025. URL <https://transformer-circuits.pub/2025/attribution-graphs/methods.html>.

566

567 John W. Backus. The syntax and semantics of the proposed international algebraic language of
 568 the zurich ACM-GAMM conference. In *Information Processing, Proceedings of the 1st Inter-
 569 national Conference on Information Processing, UNESCO, Paris 15-20 June 1959*, pp. 125–131.
 570 UNESCO (Paris), 1959.

571

572 David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection:
 573 Quantifying interpretability of deep visual representations. In *IEEE Conference on Computer
 Vision and Pattern Recognition (CVPR)*, pp. 3319–3327. IEEE Computer Society, 2017. doi:
 574 10.1109/CVPR.2017.354. URL <https://doi.org/10.1109/CVPR.2017.354>.

575

576 Steven Bills, Nick Cammarata, Dan Mossing, Henk Tillman, Leo Gao, Gabriel Goh, Ilya
 577 Sutskever, Jan Leike, Jeff Wu, and William Saunders. Language models can explain
 578 neurons in language models. <https://openaipublic.blob.core.windows.net/neuron-explainer/paper/index.html>, 2023.

579

580 Joseph Bloom. Open source sparse autoencoders for all residual stream layers of
 581 gpt2 small. <https://www.alignmentforum.org/posts/f9EgflSurAiqRJySD/open-source-sparse-autoencoders-for-all-residual-stream>, 2024.

582

583 Angie Boggust, Hyemin Bang, Hendrik Strobelt, and Arvind Satyanarayan. Abstraction alignment:
 584 Comparing model-learned and human-encoded conceptual relationships. In *Proceedings of the
 585 CHI Conference on Human Factors in Computing (CHI)*, pp. 417:1–417:20. ACM, 2025. doi:
 586 10.1145/3706598.3713406. URL <https://doi.org/10.1145/3706598.3713406>.

587

588 Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Con-
 589 erly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu,
 590 Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Zac Hatfield-Dodds, Alex
 591 Tamkin, Karina Nguyen, Brayden McLean, Josiah E Burke, Tristan Hume, Shan Carter,
 592 Tom Henighan, and Christopher Olah. Towards monosemanticity: Decomposing language
 593 models with dictionary learning. *Transformer Circuits Thread*, 2023. <https://transformer-circuits.pub/2023/monosemantic-features/index.html>.

594 Bart Bussmann, Noa Nabeshima, Adam Karvonen, and Neel Nanda. Learning multi-level features
 595 with matryoshka sparse autoencoders. *CoRR*, abs/2503.17547, 2025. doi: 10.48550/ARXIV.
 596 2503.17547. URL <https://doi.org/10.48550/arXiv.2503.17547>.

597

598 Shan Carter, Zan Armstrong, Ludwig Schubert, Ian Johnson, and Chris Olah. Activation atlas.
 599 *Distill*, 2019. doi: 10.23915/distill.00015. <https://distill.pub/2019/activation-atlas>.

600 David Chanin, James Wilken-Smith, Tomás Dulka, Hardik Bhatnagar, and Joseph Bloom. A
 601 is for absorption: Studying feature splitting and absorption in sparse autoencoders. *CoRR*,
 602 abs/2409.14507, 2024. doi: 10.48550/ARXIV.2409.14507. URL <https://doi.org/10.48550/arXiv.2409.14507>.

603

604 Noam Chomsky. Three models for the description of language. *IRE Trans. Inf. Theory*, 2(3):
 605 113–124, 1956. doi: 10.1109/TIT.1956.1056813. URL <https://doi.org/10.1109/TIT.1956.1056813>.

606

607 Juliet Corbin and Anselm Strauss. *Basics of the Qualitative Research: Techniques and Procedures
 608 for Developing Grounded Theory*. SAGE Publications, 1998. doi: 10.4135/9781452230153.

609

610 Jacob Dunefsky, Philippe Chlenski, and Neel Nanda. Transcoders find interpretable LLM
 611 feature circuits. In *Advances in Neural Information Processing Systems (NeurIPS)*,
 612 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/2b8f4db0464cc5b6e9d5e6bea4b9f308-Abstract-Conference.html.

613

614 Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna
 615 Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse,
 616 Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg, and Christopher Olah.
 617 Toy models of superposition. *Transformer Circuits Thread*, 2022. https://transformer-circuits.pub/2022/toy_model/index.html.

618

619

620 Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason
 621 Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The pile:
 622 An 800gb dataset of diverse text for language modeling. *CoRR*, abs/2101.00027, 2021. URL
 623 <https://arxiv.org/abs/2101.00027>.

624

625 Leo Gao, Tom Dupré la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya
 626 Sutskever, Jan Leike, and Jeffrey Wu. Scaling and evaluating sparse autoencoders. In *International
 627 Conference on Learning Representations (ICLR)*. OpenReview.net, 2025. URL
 628 <https://openreview.net/forum?id=tcsZt9ZNDK>.

629

630 Aaron Gokaslan, Vanya Cohen, Ellie Pavlick, and Stefanie Tellex. Openwebtext corpus. <http://Skylion007.github.io/OpenWebTextCorpus>, 2019.

631

632 Onkar Gujral, Mihir Bafna, Eric Alm, and Bonnie Berger. Sparse autoencoders uncover biologically
 633 interpretable features in protein language model representations. *Proceedings of the National
 634 Academy of Sciences*, 122(34):e2506316122, 2025. doi: 10.1073/pnas.2506316122.

635

636 Yoav Gur-Arieh, Roy Mayan, Chen Agassy, Atticus Geiger, and Mor Geva. Enhancing automated
 637 interpretability with output-centric feature descriptions. In *Proceedings of the Annual Meeting of
 638 the Association for Computational Linguistics (ACL)*, pp. 5757–5778. Association for Computational
 639 Linguistics, 2025. URL <https://aclanthology.org/2025.acl-long.288/>.

640

641 Evan Hernandez, Sarah Schwettmann, David Bau, Teona Bagashvili, Antonio Torralba, and Jacob
 642 Andreas. Natural language descriptions of deep visual features. In *International Conference
 643 on Learning Representations (ICLR)*. OpenReview.net, 2022. URL <https://openreview.net/forum?id=NudBMY-tzDr>.

644

645 Jing Huang, Atticus Geiger, Karel D’Oosterlinck, Zhengxuan Wu, and Christopher Potts.
 646 Rigorously assessing natural language explanations of neurons. In *Proceedings of the
 647 BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP (BlackboxNLP@EMNLP)*, pp. 317–331. Association for Computational Linguistics, 2023. doi: 10.
 648 18653/V1/2023.BLACKBOXNLP-1.24. URL <https://doi.org/10.18653/v1/2023.blackboxnlp-1.24>.

648 Robert Huben, Hoagy Cunningham, Logan Riggs Smith, Aidan Ewart, and Lee Sharkey. Sparse
 649 autoencoders find highly interpretable features in language models. In *International Conference*
 650 *on Learning Representations (ICLR)*. OpenReview.net, 2024. URL <https://openreview.net/forum?id=F76bwRSLeK>.
 651

652 Mingyu Jin, Qinkai Yu, Jingyuan Huang, Qingcheng Zeng, Zhenting Wang, Wenyue Hua, Haiyan
 653 Zhao, Kai Mei, Yanda Meng, Kaize Ding, Fan Yang, Mengnan Du, and Yongfeng Zhang.
 654 Exploring concept depth: How large language models acquire knowledge and concept at dif-
 655 ferent layers? In *Proceedings of the International Conference on Computational Linguis-
 656 tics (COLING)*, pp. 558–573. Association for Computational Linguistics, 2025. URL <https://aclanthology.org/2025.coling-main.37/>.
 657

658 Donald E. Knuth. On the translation of languages from left to right. *Inf. Control.*, 8(6):607–
 659 639, 1965. doi: 10.1016/S0019-9958(65)90426-2. URL [https://doi.org/10.1016/S0019-9958\(65\)90426-2](https://doi.org/10.1016/S0019-9958(65)90426-2).
 660

661 Laura Kopf, Nils Feldhus, Kirill Bykov, Philine Lou Bommer, Anna Hedström, Marina M.-C.
 662 Höhne, and Oliver Eberle. Capturing polysemy with PRISM: A multi-concept feature de-
 663 scription framework. *CoRR*, abs/2506.15538, 2025. doi: 10.48550/ARXIV.2506.15538. URL
 664 <https://doi.org/10.48550/arXiv.2506.15538>.
 665

666 Mark V. Lawson. Finite automata. In Dimitrios Hristu-Varsakelis and William S. Levine (eds.),
 667 *Handbook of Networked and Embedded Control Systems*, pp. 117–144. Birkhäuser, 2005.
 668

669 Tom Lieberum, Senthooran Rajamanoharan, Arthur Conmy, Lewis Smith, Nicolas Sonnerat, Vikrant
 670 Varma, János Kramár, Anca D. Dragan, Rohin Shah, and Neel Nanda. Gemma scope: Open sparse
 671 autoencoders everywhere all at once on gemma 2. *CoRR*, abs/2408.05147, 2024. doi: 10.48550/
 672 ARXIV.2408.05147. URL <https://doi.org/10.48550/arXiv.2408.05147>.
 673

674 Johnny Lin. Neuronpedia: Interactive reference and tooling for analyzing neural networks, 2023.
 675 URL <https://www.neuronpedia.org>. Software available from neuronpedia.org.
 676

677 Jack Lindsey, Wes Gurnee, Emmanuel Ameisen, Brian Chen, Adam Pearce, Nicholas L. Turner,
 678 Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, Jonathan Marcus, Michael Sklar, Adly
 679 Templeton, Trenton Bricken, Callum McDougall, Hoagy Cunningham, Thomas Henighan, Adam
 680 Jermyn, Andy Jones, Andrew Persic, Zhenyi Qi, T. Ben Thompson, Sam Zimmerman, Kelley
 681 Rivoire, Thomas Conerly, Chris Olah, and Joshua Batson. On the biology of a large language
 682 model. *Transformer Circuits Thread*, 2025. URL <https://transformer-circuits.pub/2025/attribution-graphs/biology.html>.
 683

684 Zachary C. Lipton. The mythos of model interpretability. *Commun. ACM*, 61(10):36–43, 2018. doi:
 685 10.1145/3233231. URL <https://doi.org/10.1145/3233231>.
 686

687 Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent
 688 Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussonot,
 689 Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose
 690 Slone, Amélie Héliou, Andrea Tacchetti, Anna Bulanova, Antonia Paterson, Beth Tsai, Bobak
 691 Shahriari, Charline Le Lan, Christopher A. Choquette-Choo, Clément Crepy, Daniel Cer, Daphne
 692 Ippolito, David Reid, Elena Buchatskaya, Eric Ni, Eric Noland, Geng Yan, George Tucker,
 693 George-Cristian Muraru, Grigory Rozhdestvenskiy, Henryk Michalewski, Ian Tenney, Ivan Gr-
 694 ishchenko, Jacob Austin, James Keeling, Jane Labanowski, Jean-Baptiste Lespiau, Jeff Stanway,
 695 Jenny Brennan, Jeremy Chen, Johan Ferret, Justin Chiu, and et al. Gemma: Open models based
 696 on gemini research and technology. *CoRR*, abs/2403.08295, 2024. doi: 10.48550/ARXIV.2403.
 697 08295. URL <https://doi.org/10.48550/arXiv.2403.08295>.
 698

699 Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. doi:
 700 10.23915/distill.00007. <https://distill.pub/2017/feature-visualization>.
 701

702 Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and
 703 Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 2018. doi: 10.23915/
 704 distill.00010. <https://distill.pub/2018/building-blocks>.

702 Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter.
 703 Zoom in: An introduction to circuits. *Distill*, 2020. doi: 10.23915/distill.00024.001.
 704 <https://distill.pub/2020/circuits/zoom-in>.

705 Kiho Park, Yo Joong Choe, and Victor Veitch. The linear representation hypothesis and the ge-
 706 ometry of large language models. In *International Conference on Machine Learning (ICML)*.
 707 OpenReview.net, 2024. URL <https://openreview.net/forum?id=UGpGkLzwpP>.

708 Gonçalo Paulo, Alex Mallen, Caden Juang, and Nora Beroose. Automatically interpreting millions
 709 of features in large language models. *CoRR*, abs/2410.13928, 2024. doi: 10.48550/ARXIV.2410.
 710 13928. URL <https://doi.org/10.48550/arXiv.2410.13928>.

711 Bruno Puri, Aakriti Jain, Elena Golimblevskaia, Patrick Kahardipraja, Thomas Wiegand, Wojciech
 712 Samek, and Sebastian Lapuschkin. FADE: why bad descriptions happen to good features. In
 713 *Findings of the Association for Computational Linguistics (ACL)*, pp. 17138–17160. Association
 714 for Computational Linguistics, 2025.

715 Python Software Foundation. Full grammar specification. The Python Language Reference, Python
 716 3.13.7, 2025. URL <https://docs.python.org/3/reference/grammar.html>.

717 Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language
 718 models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

719 Senthooran Rajamanoharan, Tom Lieberum, Nicolas Sonnerat, Arthur Conmy, Vikrant Varma, János
 720 Kramár, and Neel Nanda. Jumping ahead: Improving reconstruction fidelity with jumprelu sparse
 721 autoencoders. *CoRR*, abs/2407.14435, 2024. doi: 10.48550/ARXIV.2407.14435. URL <https://doi.org/10.48550/arXiv.2407.14435>.

722 Tamar Rott Shaham, Sarah Schwettmann, Franklin Wang, Achyuta Rajaram, Evan Hernandez, Ja-
 723 cob Andreas, and Antonio Torralba. A multimodal automated interpretability agent. In *In-
 724 ternational Conference on Machine Learning (ICML)*. OpenReview.net, 2024. URL <https://openreview.net/forum?id=mDw42ZanmE>.

725 Xiaoqing Sun, Alessandro Stolfo, Joshua Engels, Ben Wu, Senthooran Rajamanoharan, Mrinmaya
 726 Sachan, and Max Tegmark. Dense SAE latents are features, not bugs. *CoRR*, abs/2506.15679,
 727 2025. doi: 10.48550/ARXIV.2506.15679. URL <https://doi.org/10.48550/arXiv.2506.15679>.

728 Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen,
 729 Adam Pearce, Craig Citro, Emmanuel Ameisen, Andy Jones, Hoagy Cunningham, Nicholas L
 730 Turner, Callum McDougall, Monte MacDiarmid, C. Daniel Freeman, Theodore R. Sumers,
 731 Edward Rees, Joshua Batson, Adam Jermyn, Shan Carter, Chris Olah, and Tom Henighan.
 732 Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. *Trans-
 733 former Circuits Thread*, 2024. URL <https://transformer-circuits.pub/2024/scaling-monosemanticity/index.html>.

734 Ian Tenney, Dipanjan Das, and Ellie Pavlick. BERT rediscovers the classical NLP pipeline. In *Pro-
 735 ceedings of the Conference of the Association for Computational Linguistics (ACL)*, pp. 4593–
 736 4601. Association for Computational Linguistics, 2019. doi: 10.18653/V1/P19-1452. URL
 737 <https://doi.org/10.18653/v1/p19-1452>.

738 Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelli-
 739 gence*, 267:1–38, 2019. doi: 10.1016/J.ARTINT.2018.07.007. URL <https://doi.org/10.1016/j.artint.2018.07.007>.

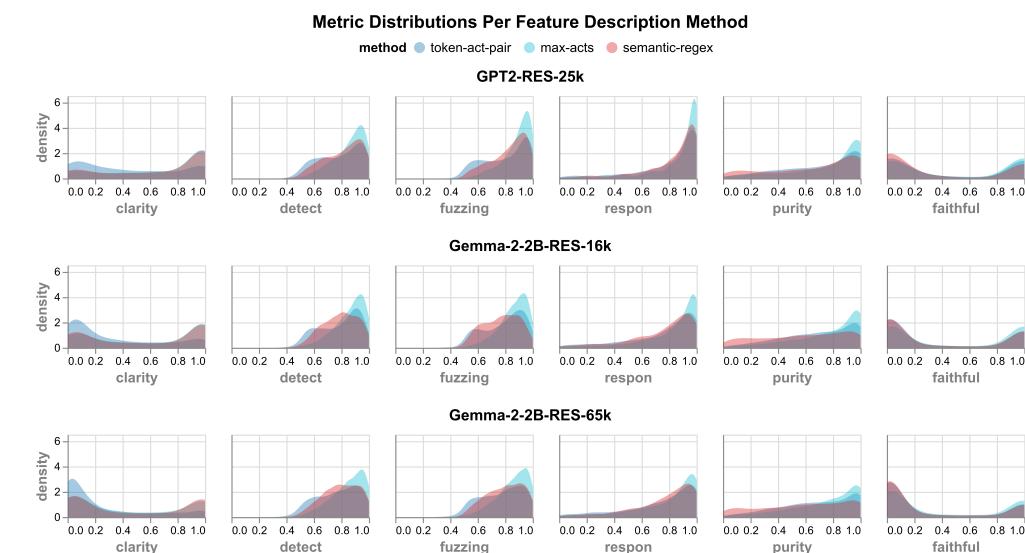
740 Guido van Rossum, Barry Warsaw, and Nick Coghlan. Style guide for Python code. PEP 8, 2001.
 741 URL <https://www.python.org/dev/peps/pep-0008/>.

742 Vladimir Zaigrajew, Hubert Baniecki, and Przemyslaw Biecek. Interpreting CLIP with hierarchical
 743 sparse autoencoders. *CoRR*, abs/2502.20578, 2025. doi: 10.48550/ARXIV.2502.20578. URL
 744 <https://doi.org/10.48550/arXiv.2502.20578>.

745

756 A LLM USAGE STATEMENT
757758 Large language models (LLMs) are the subject of this work, and they were also used as general-
759 purpose tools to assist [with writing and coding](#). All research ideas, study designs, analyses, and
760 substantive code implementations were developed by the authors.
761762 B ADDITIONAL RESULTS
763764 Here we show alternative views of the benchmarking results from Section 5.1. We show the numerical
765 means and standard deviations in Table A1 and the metric distributions in Figure A1.
766768 Table A1: Across results on GPT-2-RES-25k, Gemma-2-2B-RES-16k, and
769 Gemma-2-2B-RES-65k, semantic-regex feature descriptions perform on par [with](#) natural
770 language feature description methods. These results suggest that the semantic regex language is
771 appropriately expressive to describe features with similar performance as unconstrained natural
772 language. Each metric is computed on 100 randomly selected features per model layer and
773 displayed as the mean \pm the standard deviation.
774

	GPT-2-RES-25k	Generation		Discrimination		Faithfulness
		clarity	detection	fuzzing	responsiveness	faithfulness
	token-act-pair	0.45 \pm 0.36	0.79 \pm 0.15	0.80 \pm 0.16	0.81 \pm 0.23	0.71 \pm 0.28
	max-acts	0.70 \pm 0.35	0.86 \pm 0.12	0.88 \pm 0.10	0.87 \pm 0.20	0.78 \pm 0.26
	semantic-regex	0.68 \pm 0.36	0.81 \pm 0.14	0.83 \pm 0.13	0.83 \pm 0.21	0.66 \pm 0.32
Gemma-2-2B-RES-16k	Gemma-2-2B-RES-16k					
	token-act-pair	0.34 \pm 0.34	0.78 \pm 0.15	0.79 \pm 0.15	0.76 \pm 0.25	0.69 \pm 0.26
	max-acts	0.59 \pm 0.39	0.86 \pm 0.11	0.86 \pm 0.10	0.82 \pm 0.22	0.77 \pm 0.24
	semantic-regex	0.57 \pm 0.40	0.79 \pm 0.13	0.77 \pm 0.13	0.76 \pm 0.23	0.58 \pm 0.30
Gemma-2-2B-RES-65k	Gemma-2-2B-RES-65k					
	token-act-pair	0.25 \pm 0.32	0.77 \pm 0.15	0.77 \pm 0.15	0.75 \pm 0.26	0.69 \pm 0.26
	max-acts	0.45 \pm 0.40	0.85 \pm 0.12	0.85 \pm 0.12	0.79 \pm 0.23	0.75 \pm 0.25
	semantic-regex	0.47 \pm 0.42	0.79 \pm 0.13	0.79 \pm 0.13	0.76 \pm 0.22	0.59 \pm 0.31

806 Figure A1: Metric distributions comparing semantic-regex feature descriptions against
807 natural language baseline methods. Results are shown for 100 feature per layer on GPT-2-RES-25k,
808 Gemma-2-2B-RES-16k, and Gemma-2-2B-RES-65k and visualized using kernel density estimation
809 with bandwidth estimated via Scott’s rule.

810 C ABLATION STUDY

813 Our experiments (Section 5) use `GPT-4o-mini` as both the *explainer* and *evaluator* model. This
 814 choice follows common practice in automated interpretability, where the same model is used to gen-
 815 erate and evaluate feature descriptions (Bills et al., 2023; Gur-Arieh et al., 2025). Since verification
 816 is typically easier than generation, a model can reliably judge its descriptions and the independence
 817 of model calls prevents conditioning effects.

818 To assess whether our findings depend on the choice of model, we conduct an ablation using a more
 819 capable model (`GPT-4o`) in place of `GPT-4o-mini`. This ablation tests whether the relative perfor-
 820 mance between semantic regexes and natural language descriptions is sensitive to model capability
 821 and evaluates the robustness of our results.

822 Our ablation study follows the same automated interpretability pipeline described in Section 4. All
 823 components of the pipeline remain unchanged, including feature extraction, activation formatting,
 824 and prompting strategy. The only difference is that `GPT-4o` is substituted for `GPT-4o-mini` in both
 825 the explanation and evaluation steps. We apply this setup to a randomly sampled subset of 520
 826 features from across layers of `GPT-2-RES-25k`. For each feature, we generate descriptions using
 827 `semantic-regex`, `max-acts`, and `token-act-pair` using both `GPT-4o` and `GPT-4o-mini`, and
 828 we evaluate each description using the same discrimination and generation metrics used in the main
 829 experiments. This design isolates the effect of model capability while preserving all other aspects
 830 of the experimental pipeline.

831 We report the results in Figures A2 and A3 and Table A2. Across all feature description methods
 832 and evaluation metrics, the overall results remains the same regardless of whether the explainer
 833 and evaluator are `GPT-4o` or `GPT-4o-mini`. Semantic regexes continue to match the performance
 834 of natural language descriptions, and the relative differences between description types are nearly
 835 identical across the two model settings. While some metrics are slightly higher under `GPT-4o` (e.g.,
 836 slight improvements in `purity` and `responsiveness`) these gains arise uniformly across methods
 837 and are well within the experimental noise.

838 Overall, the consistency between `GPT-4o-mini` and `GPT-4o` demonstrates that the comparative
 839 performance of semantic regexes is not sensitive to the choice of model. Given this robustness,
 840 along with the significantly improved efficiency and lower cost of `GPT-4o-mini` (Appendix D), we
 841 use `GPT-4o-mini` for all experiments reported in the main text.

842 D COST ANALYSIS

844 The cost of generating `semantic-regex` feature descriptions is comparable to the cost of prior
 845 natural language methods. Computing descriptions for all features in `GPT-2-RES-25k` would cost

846
 847
 848
 849 Table A2: Across results on `semantic-regex`, `max-acts`, and `token-act-pair` feature de-
 850 scriptions, the choice of explainer and evaluator model (either `GPT-4o` or `GPT-4o-mini`) does not
 851 change the relative performance. These results validate our use of `GPT-4o-mini` in our main ex-
 852 periments. Each metric is computed on 520 randomly selected features from `GPT-2-RES-25k` and
 853 displayed as the mean \pm the standard deviation.

<code>token-act-pair</code>	Generation clarity	Discrimination			
		<code>detection</code>	<code>fuzzing</code>	<code>responsiveness</code>	<code>purity</code>
<code>GPT-4o-mini</code>	0.45 ± 0.37	0.79 ± 0.15	0.81 ± 0.15	0.83 ± 0.22	0.73 ± 0.26
<code>GPT-4o</code>	0.59 ± 0.37	0.83 ± 0.13	0.82 ± 0.13	0.90 ± 0.17	0.84 ± 0.21
<code>max-acts</code>					
<code>GPT-4o-mini</code>	0.71 ± 0.36	0.86 ± 0.11	0.89 ± 0.10	0.88 ± 0.20	0.81 ± 0.24
<code>GPT-4o</code>	0.73 ± 0.33	0.86 ± 0.09	0.85 ± 0.09	0.91 ± 0.14	0.86 ± 0.19
<code>semantic-regex</code>					
<code>GPT-4o-mini</code>	0.69 ± 0.37	0.81 ± 0.13	0.83 ± 0.12	0.84 ± 0.21	0.68 ± 0.32
<code>GPT-4o</code>	0.68 ± 0.36	0.82 ± 0.12	0.81 ± 0.12	0.84 ± 0.22	0.74 ± 0.30

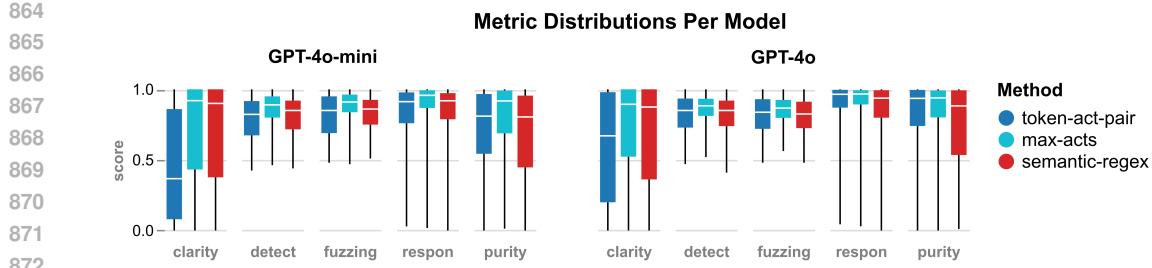


Figure A2: Metric box plots comparing different explainer and evaluator models (GPT-4o and GPT-4o-mini) across semantic-regex and natural language baseline methods (max-acts and token-act-pair). Results are shown for 520 features from GPT-2-RES-25k.

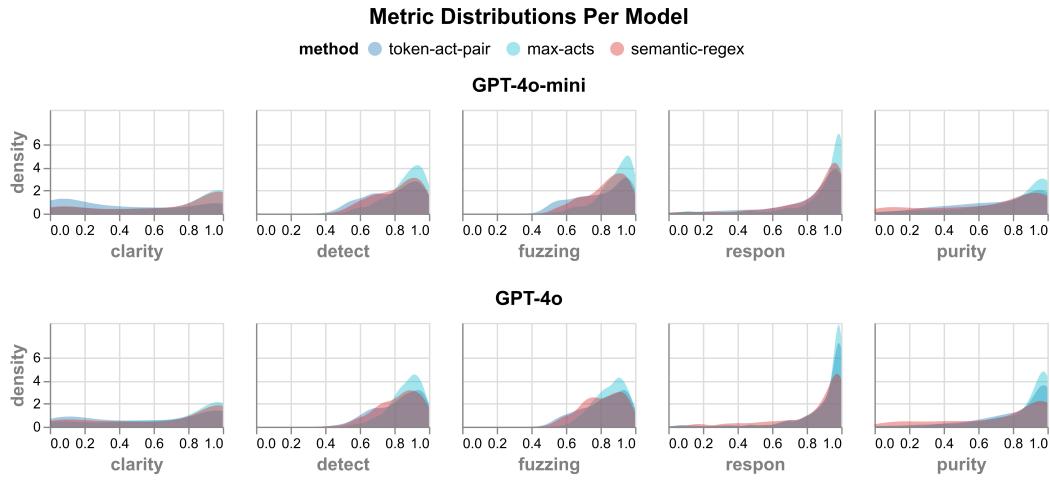


Figure A3: Metric distributions comparing different explainer and evaluator models (GPT-4o and GPT-4o-mini) across semantic-regex and natural language baseline methods. Results are shown for 520 features from GPT-2-RES-25k and visualized using kernel density estimation with bandwidth estimated via Scott’s rule.

approximately \$65.27 using GPT-4o-mini and \$1,087.86 using GPT-4o. These values are on par with to existing methods: generating all token-act-pair descriptions would cost \$67.67, and all max-acts descriptions would cost \$54.01 using GPT-4o-mini.

The description cost per feature depends on the number of input tokens in the prompt (system prompt and few-shot examples, T_{prompt} and the feature’s tokens, T_{feature}) and on the number of generated output tokens, T_{out} .

$$\text{Cost per feature} = P_{\text{in}}(T_{\text{prompt}} + T_{\text{feature}}) + P_{\text{out}}(T_{\text{out}}), \quad (1)$$

where P_{in} and P_{out} are the API prices for input and output tokens.

We report the token counts and resulting costs for each feature description method in Table A3. Differences across methods are the result of formatting choices and the number of examples shown to the explainer. For instance, the token-act-pair uses more verbose activation formatting than max-acts and semantic-regex, and semantic-regex uses fewer activating examples than max-acts but adds the semantic regex definition to the system prompt.

At the time of writing, the OpenAI API pricing² is \$0.15/\$0.60 USD per million input/output tokens for GPT-4o-mini and \$2.50/\$10.00 USD per million input/output tokens for GPT-4o. These esti-

²<https://platform.openai.com/docs/pricing>

918
 919 Table A3: Cost analysis for feature description generation. **Costs are computed from** the number
 920 of input tokens (system prompt, few-shot examples, and feature input) and **the number of** generated
 921 output tokens. We report the mean \pm std of input and output **token counts** across 130 features from
 922 GPT-2-RES-25k (10 per layer). Tokenization is performed using OpenAI’s **GPT-4o** model family
 923 tokenizer.

	token-act-pair	max-acts	semantic-regex
Input Tokens: system and few-shot examples (T_{prompt})	919	483	993
Input Tokens: average per feature (T_{feature})	457 ± 213	524 ± 90	237 ± 47
Output Tokens: average per generation (T_{out})	9 ± 3	30 ± 7	33 ± 12
GPT-4o-mini cost per feature	\$ 0.00021180	\$ 0.00016905	\$ 0.00020430
GPT-4o cost per feature	\$ 0.00353000	\$ 0.00281750	\$ 0.00340500

930
 931 **mates** assume no input caching, so the reported prices are an upper bound on the total **cost of feature**
 932 **description generation**.

934 E USER STUDY PROTOCOL

935 Here we provide the full protocol for our user study in Section 5.4.

936 Since we already evaluate semantic regex reliability across thousands of features in Section 5.1, the
 937 purpose of our user study is not to obtain another quantitative reliability estimate. Instead, our goal
 938 is to gain qualitative insight into how semantic regexes shape real-world AI experts’ interpretation
 939 of model features. To do this effectively, we recruit a representative sample of the practitioners
 940 most likely to use semantic regexes in practice and present them with a controlled but diverse set
 941 of model features. We intentionally keep the study size at 24 experts and 12 features to make the
 942 protocol feasible. Importantly, we find that these sample sizes are sufficient to draw insights about
 943 the benefits and limitations of semantic regexes.

944
 945 **Participants** Our target population consists of practitioners who would realistically interpret se-
 946 mantic regex feature descriptions in their real-world tasks, including interpretability research, AI
 947 safety, and model development. Therefore, we recruited 24 AI experts from within our organization
 948 who work with LLMs. These participants have a broad range of technical roles and backgrounds,
 949 ensuring that the study captures how semantic regexes are interpreted by the types of experts who
 950 would use them in practice.

951
 952 **Features** Our user study investigates how the *format* of a feature description influences people’s
 953 understanding. To ensure that participants interpreted meaningful feature descriptions, we restricted
 954 to features from GPT-2-RES-25k with both accurate semantic regex and natural language descrip-
 955 tions. Because our goal was not to exhaustively evaluate all such features, but rather to study human
 956 interpretation across a representative sample, we focused on selecting a diverse subset.

957
 958 We first filtered to features whose *semantic-regex* and *max-acts* descriptions each achieved
 959 a detection score above 75%. From this filtered set, we manually selected 12 features whose
 960 descriptions we verified to be accurate. Our selection process emphasized feature diversity by se-
 961 lecting features varying depths and with a range of activation patterns. We selected features from
 962 across the model’s layers, resulting in 5 features from early layers (1-4), 4 from middle layers (5-8),
 963 and 3 from late layers (9-12). And, we chose features that represented diverse activation patterns,
 964 such that 2 activated on exact *symbols*, 1 on a *lexeme*, 3 on a *field*, 2 only activated in a particu-
 965 lar *context*, and 4 were complex patterns that required concatenation or combination of semantic
 966 *regex components*.

967
 968 This selection process allowed us to study human interpretation across a controlled but diverse range
 969 of features without overburdening participants. We show all selected features and their descriptions
 970 in Figure A4.

971
 972 **Task** Participants were tasked with writing phrases that matched or were a close counterfactual
 973 to a given feature description. Each participant was shown three natural language descriptions and
 974 three semantic regex descriptions. To avoid carryover effects, participants saw natural language and

972 semantic regex descriptions from different features. To avoid learning effects, we randomized the
973 order, with half the participants seeing natural language descriptions first and the other half seeing
974 semantic regexes first.

975 Each participant completed the task for three natural language feature descriptions (max-acts)
976 and three semantic-regex feature descriptions. We showed each participant features of varying
977 complexity, with half the participants seeing the feature descriptions in the left column of Figure A4
978 and the other half seeing feature descriptions from the right column.

979 The task was completed asynchronously and distributed via a survey link. The survey began with an
980 introduction to the task, the semantic regex language, and an example of matching and counterfactual
981 phrases (Figure A5). Then, participants were shown an explanation of the feature description type
982 they would see first, followed by three feature descriptions they were asked to generate phrases
983 for. This was repeated with the alternate feature description type and the final three explanations
984 (Figures A6 and A7). For each feature description, they were asked to “Write 3 phrases/sentences
985 that match the feature description” and “Write 1 phrase/sentence that is a counterfactual for the
986 semantic regex feature description” (Figures A8 and A9).

987

988

989

990

991

992

993

994

995

996

997

998

999

1000

1001

1002

1003

1004

1005

1006

1007

1008

1009

1010

1011

1012

1013

1014

1015

1016

1017

1018

1019

1020

1021

1022

1023

1024

1025



Figure A4: The features we used in our user study (Section 5.4), displayed as their top activating examples and semantic-regex and max-acts feature descriptions. Half of the participants were shown the features in the left column and the other half was shown features in the right column.

1080

1081

1082

1083

1084

1085

1086

1087

Welcome!

This survey is part of a research project on LLM feature descriptions. It should take **~10 minutes**. Thank you for taking part!

Task Overview

Your job is to write phrases/sentences based on LLM feature descriptions.

For each feature description, you will write:

- 3 phrases/sentences that **match** the description. The goal is that these phrases activate the LLM feature.
- 1 phrase/sentence that is a **counterfactual** for the description (i.e., it almost matches the description but doesn't). The goal is that these phrases do not activate the LLM feature.

You will get **3 natural language** feature descriptions and **3 semantic regex** feature descriptions.

Semantic Regex Format

A **semantic regex** is a structured language pattern.

Basic Elements:

- **[:symbol X:]** → matches X
 - **[:symbol color:]** matches "image in color"
- **[:lexeme X:]** → matches X and its syntactic variants
 - **[:lexeme color:]** matches color, colors, coloring, colored, etc. such as "coloring book" and "many colors"
- **[:field X:]** → matches semantic variants of X
 - **[:field color:]** matches red, orange, blue, etc. such as "blue skies"

These elements can be combined together:

- **@{:context X}(A)** → matches semantic regex A when it appears in context X
 - **@{:context politics}[:field color:]** matches "a blue state" but not "blue skies"
- **[A][B]** → matches semantic regex A immediately before semantic regex B
 - **[:lexeme color:][:field color:]** matches "the color blue" and "colored yellow"
- **[A][B]** → matches semantic regex A OR semantic regex B
 - **[:field color:] [:symbol and:][:symbol or:] [:field color:]** matches "red or blue" or "pink and purple"

Example

Natural Language Feature Description: Mentions of time durations associated with prison sentences, highlighting various lengths ranging from months to years.

Semantic Regex Feature Description: **@{:context prison sentences}[:field duration:]**

Matching Phrases:

1. sentenced to 30 years in prison
2. spent four years behind bars
3. 7 months of jail time

Counterfactual Phrase:

1. spent four years living in California

Your responses are voluntary and you may stop at any time.

Figure A5: The user study introduction **explains the task, the semantic regex language, and provides an example**.

1128

1129

1130

1131

1132

1133

1134
 1135
 1136
 1137
 1138 You will start with **3 semantic regex** feature descriptions. As a reminder:
 1139
 1140 A **semantic regex** is a structured language pattern.
 1141 Basic Elements:
 1142 • [:symbol X:] → matches X
 1143 • [:lexeme X:] → matches X and its syntactic variants
 1144 • [:lexeme color:] matches color, colors, coloring, colored, etc. such as "coloring book" and "many colors"
 1145 • [:field X:] → matches semantic variants of X
 1146 • [:field color:] matches red, orange, blue, etc. such as "blue skies"
 1147 These elements can be combined together:
 1148 • @{:context X}(A) → matches semantic regex A when it appears in context X
 1149 • @{:context politics}[:field color:] matches "a blue state" but not "blue skies"
 1150 • [A][B] → matches semantic regex A immediately before semantic regex B
 1151 • [:lexeme color] [:field color:] matches "the color blue" and "colored yellow"
 1152 • [A][B] → matches semantic regex A OR semantic regex B
 1153 • [:field color] [:symbol and:] [:symbol or:] [:field color:] matches "red or blue" or "pink and purple"
 1154
 1155 **Example**
 1156 Semantic Regex Feature Description:
 1157 @{:context prison sentences}{[:field duration:]}
 1158
 1159 Matching Phrases:
 1160 1. sentenced to 30 years in prison
 1161 2. spent four years behind bars
 1162 3. 7 months of jail time
 1163
 1164 Counterfactual Phrase:
 1165 1. spent four years living in California
 1166
 1167
 1168
 1169

Figure A6: The user study's description of semantic regexes.

1170 Now you will see **3 natural language** feature descriptions.
 1171
 1172 The feature descriptions describe data that activates an LLM feature. Aim to generate phrases/sentences where that description would
 1173 (or would not) apply.
 1174
 1175 **Example**
 1176 Natural Language Feature Description:
 1177 Mentions of time durations associated with prison sentences, highlighting various lengths ranging from months to years.
 1178
 1179 Matching Phrases:
 1180 1. sentenced to 30 years in prison
 1181 2. spent four years behind bars
 1182 3. 7 months of jail time
 1183
 1184 Counterfactual Phrase:
 1185 1. spent four years living in California
 1186
 1187

Figure A7: The user study's description of natural language feature descriptions.

1188
 1189
 1190
 1191
 1192
 1193
 1194
 1195
 1196
 1197
 1198

1199 * Write 3 phrases/sentences that **match** the *semantic regex* feature description:
 1200 **[:symbol on:]**

1201
 1202
 1203

1204 * Write 1 phrase/sentence that is a **counterfactual** for the *semantic regex* feature description:
 1205 **[:symbol on:]**

1206
 1207

1208 *Instructions Reminder:*

1209 A *semantic regex* is a structured language pattern.

1210 Basic Elements:

- **[:symbol X:]** → matches X
 - **[:symbol color:]** matches "image in color"
- **[:lexeme X:]** → matches X and its syntactic variants
 - **[:lexeme color:]** matches color, colors, coloring, colored, etc. such as "coloring book" and "many colors"
- **[:field X:]** → matches semantic variants of X
 - **[:field color:]** matches red, orange, blue, etc. such as "blue skies"

1215 These elements can be combined together:

- **@{:context X}{(A)}** → matches semantic regex A when it appears in context X
 - **@{:context politics}{[:field color:]}** matches "a blue state" but not "blue skies"
- **[A][B]** → matches semantic regex A immediately before semantic regex B
 - **[:lexeme color:][:field color:]** matches "the color blue" and "colored yellow"
- **[A][|][B]** → matches semantic regex A OR semantic regex B
 - **[:field color:] [:symbol and:][:symbol or:][:field color:]** matches "red or blue" or "pink and purple"

1220
 1221

Example

Semantic Regex Feature Description:

@{:context prison sentences}{[:field duration:]}

1223
 1224

Matching Phrases:

1. sentenced to 30 years in prison
2. spent four years behind bars
3. 7 months of jail time

1227
 1228

Counterfactual Phrase:

1. spent four years living in California

1229
 1230

1231 Figure A8: **An example of** a user study question for a semantic regex feature description.
 1232
 1233
 1234
 1235
 1236
 1237
 1238
 1239
 1240
 1241

1242
 1243
 1244
 1245
 1246
 1247
 1248
 1249
 1250
 1251
 1252
 1253
 1254
 1255
 1256
 1257

1258 ① * Write 3 phrases/sentences that **match** the *natural language* feature description:
 1259 **Frequent use of the phrase "on" in various contexts indicating relationships, events, or conditions in sentences.**

1260
 1261
 1262 ① * Write 1 phrase/sentence that is a **counterfactual** for the *natural language* feature description:
 1263 **Frequent use of the phrase "on" in various contexts indicating relationships, events, or conditions in sentences.**

1264
 1265
 1266

Instructions Reminder:

1267 The feature descriptions describe data that activates an LLM feature. Aim to generate phrases/sentences where that description would
 1268 (or would not) apply.

1269
 1270

Example

1271
 1272

Natural Language Feature Description:

Mentions of time durations associated with prison sentences, highlighting various lengths ranging from months to years.

1273

Matching Phrases:

1274 1. sentenced to 30 years in prison
 1275 2. spent four years behind bars
 1276 3. 7 months of jail time

Counterfactual Phrase:

1277 1. spent four years living in California

1278
 1279

1280 Figure A9: An example of a user study question for a natural language feature description.
 1281
 1282
 1283
 1284
 1285
 1286
 1287
 1288
 1289
 1290
 1291
 1292
 1293
 1294
 1295

1296
1297
1298
1299
1300Table A4: Hyperparameters used to generate feature descriptions using `semantic-regex` and the natural language benchmarks (`token-act-pair` (Bills et al., 2023) and `max-acts` (Paulo et al., 2024)). Benchmark hyperparameters follow Neuronpedia’s implementations of the original methods (Lin, 2023).1301
1302
1303
1304
1305
1306
1307
1308
1309

Parameter	token-act-pair	max-acts	semantic-regex
Number of few shot examples	3	3	4
Number of data examples	5	20	10
Number of tokens per example	64	32	32
Explanation model temperature	1.0	0.7	1.0
Explanation model top-p	1.0	1.0	1.0
Activation threshold	—	60%	30%

1310
1311

F IMPLEMENTATION DETAILS

1312
1313

F.1 FEATURE DESCRIPTION METHODS

We evaluate three approaches for generating natural language explanations of LLM features: `token-act-pair` (Bills et al., 2023), `max-acts` (Paulo et al., 2024), and our proposed `semantic-regex` method. All methods use `gpt-4o-mini` as the explainer model. We follow Neuronpedia’s reference implementations (Lin, 2023) for each baseline method. Implementation details for each method are provided below, and a summary of the settings is shown in Table A4.

1319
1320
1321
1322
1323
1324

token-act-pair We follow Neuronpedia’s implementation (Lin, 2023) of the feature description method proposed in Bills et al. (2023), called `oai_token-act-pair` in the Neuronpedia interface. The explainer model is prompted with the original feature description system instructions and the three few-shot examples used in their paper (Listing 2). For each feature, we supply the top five activating data examples from [Neuronpedia](#) (Lin, 2023), where each example consists of up to a 64-token window centered on the maximally activating token. Tokens are presented in the format:

1325
1326
1327
1328
1329
1330

```
<start>
token_1 activation_1
token_2 activation_2
...
<end>
```

1331
1332
1333
1334

where `activation_i` is the normalized activation value for the feature. Activations are linearly scaled between 0 and 10, such that the feature’s maximum activation value maps to 10. The prompt ends with the continuation cue “*the main thing this neuron does is find*”, following the original protocol. Generation is performed with temperature = 1 and top-p = 1.

1335
1336
1337
1338
1339
1340
1341
1342

max-acts We follow Neuronpedia’s implementation (Lin, 2023) of the feature description method proposed in Paulo et al. (2024), called `eleuther_acts_top20` in the Neuronpedia interface. The explainer model is prompted with the original system instructions and the three few-shot examples provided in the original paper (Listing 3). For each feature, we supply the top 20 activating data examples from [Neuronpedia](#) (Lin, 2023), where each example consists of up to a 32-token window centered around the maximally activating token. Tokens exceeding 60% of the feature’s maximum activation are highlighted and contiguous activating tokens are merged into a single highlighted segment, like:

1343
1344

```
token_1<<token_2token_3>>token4...
```

1345
1346
1347

The prompt ends with the instruction to “*describe the text latents that are common in the examples*”. Generation is performed with temperature = 0.7 and top-p = 1, following Neuronpedia’s settings.

1348
1349

semantic-regex To generate semantic regexes, we prompt the explainer model with system instructions adapted from `max-acts`, augmented with a concise definition of the semantic regex language (Listing 1) and four few-shot examples illustrating the range of available primitives and

1350
 1351 modifiers (Listing 4). For each feature, we supply the top 10 activating data examples from [Neu-
 1352 ronpedia](#) (Lin, 2023), where each example consists of up to a 32-token window centered on the
 1353 maximally activating token. Tokens exceeding 30% of the feature’s maximum activation are high-
 1354 lighted, and contiguous spans are merged into single segments, as in `max-acts`.
 1355

1356 `token_1<<token_2token_3>>token4...`

1357 We found this lower threshold produced more consistent highlights, which is important for mapping
 1358 activations to the structured semantic regex language. The prompt concludes with the instruction
 1359 to “*output a short explanation followed by a semantic regex*”, which we found improved syntactic
 1360 adherence. Generation is performed with temperature = 1 and top-p = 1.

1361 A Semantic Regex is a structured pattern composed of:
 1362 * [:symbol X:] - matches an exact phrase X (e.g., [:symbol running:] matches "I am<< running>>" and "<<
 running>> faster").
 1363 * [:lexeme X:] - matches a phrase X and its syntactic variants (e.g., [:lexeme run:] matches "she<< ran>>",
 "it's<< running>> quickly").
 1364 * [:field X:] - matches a phrase X and its semantic variants (e.g., [:field run:] matches "out for a << jog
 >>" and "<< sprint>> for gold").
 1365 X can be a subword (e.g., ing), word (e.g., running), or phrase (e.g., running tempo).
 1366 These components can be combined to match more complex patterns:
 1367 * S1 S2 - matches a sequence where S1 is followed by S2 (e.g., [:symbol run:] [:lexeme fast:] matches "I<<
 run fast>>" and "they<< run faster>>").
 1368 * S1|S2 - matches either S1 or S2 (e.g., [:symbol run:][:symbol walk:] matches "I<< run>>" and "I<< walk
 >>").
 1369 * S? - matches S or nothing (e.g., [:lexeme run:] [:symbol very:]? [:symbol fast:] matches "I am<< running
 fast>>" and "I<< run very fast>>").
 1370 * @(:context C:)(S) - matches S that only activates in the context C (e.g., @(:context political:)(:lexeme
 run:) matches "she<< ran>> for office" and "<<running>> for govenor" but not "I<< run>> marathons").
 1371

1372 Listing 1: The semantic-regex description that we inject into method and metric prompts.

1373
 1374
 1375 <system>
 1376 We’re studying neurons in a neural network. Each neuron looks for some particular thing in a short document
 . Look at the parts of the document the neuron activates for and summarize in a single sentence what the
 1377 neuron is looking for. Don’t list examples of words.
 1378
 1379 The activation format is `token<tab>activation`. Activation values range from 0 to 10. A neuron finding what
 1380 it’s looking for is represented by a non-zero activation value. The higher the activation value, the
 1381 stronger the match.
 1382 </system>
 1383
 1384 <user>
 1385 Neuron 1
 1386 Activations:
 1387 <start>
 1388 t 0
 1389 urt 0
 1390 ur 0
 1391 ro 0
 1392 is 0
 1393 fab 0
 1394 ulously 0
 1395 funny 0
 1396 and 0
 1397 over 0
 1398 the 0
 1399 top 0
 1400 as 0
 1401 a 0
 1402 ' 0
 1403 very 0
 1404 sneaky 0
 1405 ' 1
 1406 but 0
 1407 ler 0
 1408 who 0
 1409 excel 0
 1410 s 0
 1411 in 0
 1412 the 0
 1413 art 0
 1414 of 0
 1415 impossible 0
 1416 disappearing 6
 1417 / 0
 1418 re 0
 1419 app 0
 1420 earring 10
 1421 acts 0
 1422 <end>
 1423 <start>
 1424 esc 0
 1425 aping 9

```

1404     the 4
1405     studio 0
1406     , 0
1407     pic 0
1408     col 0
1409     i 0
1410     is 0
1411     warm 0
1412     ly 0
1413     affecting 3
1414     and 0
1415     so 0
1416     is 0
1417     this 0
1418     ad 0
1419     roit 0
1420     ly 0
1421     minimalist 0
1422     movie 0
1423     . 0
1424     <end>
1425
1426     Same activations, but with all zeros filtered out:
1427     <start>
1428     ' 1
1429     disappearing 6
1430     earing 10
1431     <end>
1432     <start>
1433     aping 9
1434     the 4
1435     affecting 3
1436     <end>
1437
1438     Explanation of neuron 1 behavior: the main thing this neuron does is find
1439     </user>
1440
1441     <assistant>
1442     present tense verbs ending in 'ing'.
1443     </assistant>
1444
1445     <user>
1446     Neuron 2
1447     Activations:
1448     <start>
1449     as 0
1450     sac 0
1451     char 0
1452     ine 0
1453     movies 0
1454     go 0
1455     , 0
1456     this 0
1457     is 0
1458     likely 0
1459     to 0
1460     cause 0
1461     massive 0
1462     cardiac 0
1463     arrest 10
1464     if 0
1465     taken 0
1466     in 0
1467     large 0
1468     doses 0
1469     . 0
1470     <end>
1471     <start>
1472     shot 0
1473     perhaps 0
1474     ' 0
1475     art 0
1476     istically 0
1477     ' 0
1478     with 0
1479     handheld 0
1480     cameras 0
1481     and 0
1482     apparently 0
1483     no 0
1484     movie 0
1485     lights 0
1486     by 0
1487     jo 0
1488     aquin 0
1489     b 0
1490     aca 0
1491     - 0
1492     as 0
1493     ay 0
1494     , 0
1495     the 0

```

```

1458
1459     low 0
1460     - 0
1460     budget 0
1461     production 0
1461     swings 0
1461     annoy 0
1462     ingly 0
1462     between 0
1463     vert 0
1464     igo 9
1464     and 0
1465     opacity 0
1466     . 0
1466     <end>
1467     Same activations, but with all zeros filtered out:
1468     <start>
1468     arrest 10
1469     <end>
1470     <start>
1470     igo 9
1471     <end>
1472     Explanation of neuron 2 behavior: the main thing this neuron does is find
1473     </user>
1474     <assistant>
1474     words related to physical medical conditions.
1475     </assistant>
1476     <user>
1477     Neuron 3
1477     Activations:
1478     <start>
1478     the 0
1479     sense 0
1480     of 0
1480     together 3
1481     ness 7
1481     in 0
1482     our 0
1482     town 1
1483     is 0
1483     strong 0
1484     . 0
1485     <end>
1485     <start>
1486     a 0
1486     buoy 0
1487     ant 0
1487     romantic 0
1487     comedy 0
1488     about 0
1488     friendship 0
1489     , 0
1489     love 0
1490     , 0
1490     and 0
1490     the 0
1491     truth 0
1491     that 0
1492     we 2
1492     're 4
1493     all 3
1493     in 7
1494     this 10
1494     together 5
1495     . 0
1495     <end>
1496     Explanation of neuron 3 behavior: the main thing this neuron does is find
1497     </user>
1498     <assistant>
1498     phrases related to community.
1499     </assistant>
1500
1501
1502
1503

```

1504 Listing 2: The token-act-pair prompt and few-shot examples. <system></system>,
1505 <user></user>, and <assistant></assistant> delineate the message roles passed to
1506 the explainer model.

```

1507
1508
1509     <system>
1510     You are a meticulous AI researcher conducting an important investigation into patterns found in language.
1510     Your task is to analyze text and provide an explanation that thoroughly encapsulates possible patterns
1511     found in it.
1511     Guidelines:

```

```

1512
1513 You will be given a list of text examples on which special words are selected and between delimiters like
1514 <<this>>. If a sequence of consecutive tokens all are important, the entire sequence of tokens will be
1515 contained between delimiters <<just like this>>.
1516
1517 - Try to produce a concise final description. Simply describe the text latents that are common in the
1518 examples, and what patterns you found.
1519 - If the examples are uninformative, you don't need to mention them. Don't focus on giving examples of
1520 important tokens, but try to summarize the patterns found in the examples.
1521 - Do not mention the marker tokens (<< >>) in your explanation.
1522 - Do not make lists of possible explanations. Keep your explanations short and concise.
1523 - The last line of your response must be the formatted explanation, using [EXPLANATION]:
1524 </system>
1525
1526 <user>
1527 Example 1: and he was <<over the moon>> to find
1528 Example 2: we'll be laughing <<till the cows come home>>! Pro
1529 Example 3: thought Scotland was boring, but really there's more <<than meets the eye>>! I'd
1530 </user>
1531
1532 <assistant>
1533 [EXPLANATION]: Common idioms in text conveying positive sentiment.
1534 </assistant>
1535
1536 <user>
1537 Example 1: a river is wide but the ocean is wid<<er>>. The ocean
1538 Example 2: every year you get tall<<er>>, " she
1539 Example 3: the hole was small<<er>> but deep<<er>> than the
1540 </user>
1541
1542 <assistant>
1543 [EXPLANATION]: The token "er" at the end of a comparative adjective describing size.
1544 </assistant>
1545
1546 <user>
1547 Example 1: something happening inside my <<house>>", he
1548 Example 2: presumably was always contained in <<a box>>", according
1549 Example 3: people were coming into the <<smoking area>>.
1550
1551 However he
1552 Example 4: Patrick: "why are you getting in the << way?>>" Later,
1553 </user>
1554
1555 <assistant>
1556 [EXPLANATION]: Nouns representing a distinct objects that contains something, sometimes preceding a
1557 quotation mark.
1558 </assistant>

```

1540
1541 Listing 3: The max-acts prompt and few-shot examples. <system></system>,
1542 <user></user>, and <assistant></assistant> delineate the message roles passed to
1543 the explainer model.

```

1544
1545 <system>
1546 You are interpreting the role of LLM features. Your task is to describe patterns across activating text
1547 examples.
1548
1549 Input:
1550 You will be given a list of text examples.
1551 Activating phrases in each example are highlighted between delimiters like<< this and that>>.
1552
1553 Output:
1554 You will output a **Semantic Regex** that describes patterns across the text examples.
1555 A Semantic Regex is a structured pattern composed of:
1556 * [:symbol X:] - matches an exact phrase X (e.g., [:symbol running:] matches "I am<< running>>" and "<<
running>> faster").
1557 * [:lexeme X:] - matches a phrase X and its syntactic variants (e.g., [:lexeme run:] matches "she<< ran>>", "it's<<
running>> quickly").
1558 * [:field X:] - matches a phrase X and its semantic variants (e.g., [:field run:] matches "out for a << jog
>>" and "<< sprint>> for gold").
1559 X can be a subword (e.g., ing), word (e.g., running), or phrase (e.g., running tempo).
1560 These components can be combined to match more complex patterns:
1561 * S1 S2 - matches a sequence where S1 is followed by S2 (e.g., [:symbol run:] [:lexeme fast:] matches "I<<
run fast>>" and "they<< run faster>>").
1562 * S1|S2 - matches either S1 or S2 (e.g., [:symbol run:][:symbol walk:] matches "I<< run>>" and "I<< walk
>>").
1563 * S? - matches S or nothing (e.g., [:lexeme run:] [:symbol very:]? [:symbol fast:] matches "I am<< running
fast>>" and "I<< run very fast>>").
1564 * @(:context C:)(S) - matches S that only activates in the context C (e.g., @(:context political:)([:lexeme
run:]) matches "she<< ran>> for office" and "<<running>> for govenor" but not "I<< run>> marathons").
1565
1566 Instructions:
1. Look at the text examples to identify patterns that occur across **all** examples.
2. First, look for patterns within the << >> delimiters.
3. If you find an exact phrase, use a [:symbol X:].
4. If you find a phrase and its syntactic variants, use a [:lexeme X:].
5. If you find a phrase and its semantic variants, use a [:field X:].
6. Create a Semantic Regex (S) with the fewest components that precisely describes the pattern.
7. Next, look for patterns in the examples' topics.

```

```

1566
1567     1. If all examples are related to the same topic (C) AND the topic is not redundant with the current
1568        Semantic Regex (S), use a @{:context C:}(S) modifier.
1569     4. Output the simplest and most concise Semantic Regex that precisely describes the patterns across all
1570        examples.
1571     5. Do not include the delimiters tokens (<< >>) in your Semantic Regex.
1572     6. Output a short explanation followed by "SR: " and then the Semantic Regex
1573     </system>
1574
1575     <user>
1576     1: ax=[figg.add_subplot(2,1,k+)<< for>> k in xrange(2) ]
1577     2: p = 0<< for>> q in qlist: pprev = p
1578     3: << for>> lam, prob in suite.items():
1579     </user>
1580
1581     <assistant>
1582     The phrase 'for' activates only in the context of coding. SR: @{:context coding:}(:symbol for:)
1583     </assistant>
1584
1585     <user>
1586     1: extradition legislation<< prohibits an individual from being sent back>>
1587     2: << whether Israel can lift the collective protection of asylum seekers>>
1588     3: << called on Dutch>> authorities<< to do more to protect human rights>> workers
1589     </user>
1590
1591     <assistant>
1592     The activating phrases and their surrounding phrases are all related to human rights legislation. SR: [:field
1593     human rights legislation:]
1594     </assistant>
1595
1596     <user>
1597     1: off the markets - 2<< times the size>> of India's
1598     2: needed to meet demand - seven times<< the existing>> number
1599     3: That's about three times<< the>> rate seen in Hawaii
1600     </user>
1601
1602     <assistant>
1603     The symbol 'the' activates when it is between a numerical multiplier and an amount of something. SR: [:field
1604     number:] [:symbol times the] [:field amount:]
1605     </assistant>
1606
1607     <user>
1608     1: sentenced to<< 30 years in>> prison
1609     2: to<< ten months>> in prison suspended for<< 3 years>>
1610     3: << 1 month>> behind bars but was ordered to
1611     </user>
1612
1613     <assistant>
1614     The activating phrases are durations of prison sentences. SR: @{:context prison sentences}(:field duration
1615     :]
1616     </assistant>

```

1596 Listing 4: The semantic-regex prompt and few-shot examples. <system></system>, 1597 <user></user>, and <assistant></assistant> delineate the message roles passed to 1598 the explainer model.

F.2 EVALUATION METRICS

1603 We evaluate the feature description metrics across a suite of evaluation metrics. We take these 1604 evalution metrics from Paulo et al. (2024) (Eleuther) and Puri et al. (2025) (FADE). We follow their 1605 implementations closely, but make some modifications for consistency across metrics. To account 1606 for the semantic regex structure, we make small adjustments to the original prompts and inject a 1607 description of the semantic regex language (full prompts in Listings 5, 6, 7, 8, 9, 10, 11 and 12). 1608 The hyperparameters are listed in Table A5.

1609 All metrics use gpt-4o-mini as the evaluator model. Metrics typically rely on a set of activating 1610 examples (positives) and random examples (negatives). Unless otherwise specified, we use 50 1611 activating examples and 50 negative examples per feature, each consisting of a 32-token window 1612 centered on the maximally activating token. Activating example sampling varies per metric, but all 1613 random examples are sampled from alternative features in Neuronpedia; given the number and 1614 diversity of features, this approximates random dataset sampling. To evaluate the description, many of 1615 the methods compute the feature's activation on these data sets. Following prior work (Paulo et al., 1616 2024), when we compute activations, we ignore the beginning-of-sequence tokens.

F.2.1 ELEUTHER METRICS

1617 We implement the detection and fuzzing metrics from Paulo et al. (2024), following Neuronpe- 1618 dia's implementation (eleuther_recall and eleuther_fuzz). For Eleuther metrics, activating

1620

1621 Table A5: The hyperparameters used to evaluate feature descriptions using metrics from Paulo
1622 et al. (2024) (detection, fuzzing) and Puri et al. (2025) (clarity, responsiveness, purity,
1623 faithfulness).

Parameter	clarity	detection	fuzzing	responsiveness	purity	fuzzing
Number of positive examples	50	50	50	50	50	—
Number of random examples	50	50	50	50	50	10
Number of tokens per example	32	32	32	32	32	32
Number of examples per model call	—	5	5	15	15	15
Positive example sampling method	—	quantiles	quantiles	percentile	percentile	—
Number of quantiles	—	10	10	—	—	—
Percentiles	—	—	—	0, 50, 75, 95, 100	0, 50, 75, 95, 100	—
Top sampling percentage	—	—	—	20%	20%	—
Evaluation model temperature	1.0	0.7	0.7	1.0	1.0	1.0
Number of generation runs	10	—	—	—	—	—
Modification factors	—	—	—	—	—	0, 1, 10, 100
Number of steered generation tokens	—	—	—	—	—	30

1633

1634 examples are sampled uniformly across 10 activation quantiles. Across both metrics the evaluator
1635 model is tasked with providing a binary judgment of whether the each example matches the fea-
1636 ture description. The evaluator model processes 5 examples per call, with temperature = 0.7 and a
1637 maximum of 500 completion tokens.

1638

1639 **detection** The detection metric evaluates description quality at the example level. The eval-
1640 uator model prompts are shown in Listings 5 and 6. The evaluator is shown a feature description
1641 and an example and asked whether the description matches the example’s text. The final score is the
1642 balanced accuracy of these binary judgments compared to the ground-truth example labels (positive
1643 = activating, negative = random).

1644

1645 **fuzzing** The fuzzing metric follows the same setup as detection but evaluates matches at the
1646 activation level. In each of the examples, we highlight activating tokens, and the evaluator model
1647 is asked whether the description matches the highlighted regions. The evaluator model prompts
1648 are shown in Listings 7 and 8. For activating examples, we highlight tokens that activate higher
1649 than the activation threshold (60% of the feature’s maximum activation for max-acts and 30% of
1650 the feature’s maximum activation for semantic-regex) and merge contiguously activating tokens,
1651 e.g.:

1652

1653 token_1<<token_2token_3>>token_4...

1654

1655 Random examples are highlighted using the activation pattern of the positive examples, which en-
1656 sures an equal distribution of activating tokens across both example sets. The evaluator is asked
1657 whether the description matches the highlighted regions, and the score is computed as the balanced
1658 accuracy of these judgments relative to the ground-truth labels. This yields a stricter variant of
1659 detection by focusing the match decision on the most activating regions.

1660

F.2.2 FADE METRICS

1661

1662 We implement the clarity, responsiveness, purity, and faithfulness metrics from Puri
1663 et al. (2025), adapting their setup for consistency with the Eleuther metrics. We sample activating
1664 examples following FADE’s stratified sampling protocol: 20% from the top activations and the
1665 remainder sampled uniformly across the percentile bins [0, 50), [50, 75), [75, 95), and [95, 100].

1666

1667 **responsiveness and purity** To compute responsiveness and purity, the evaluator model
1668 is shown a feature description and a data example and asked to provide a discrete judgment (0, 1,
1669 or 2) indicating how strongly the text matches the description. Following FADE, samples labeled
1670 as 1 are discarded before scoring. The evaluator model prompts are shown in Listings 9 and 10.
1671 We parallelize scoring, such that the evaluator judges 15 examples at once. responsiveness
1672 computes the Gini index of the resulting scores, capturing how well the description’s matches rank-
1673 order the data by activation strength. purity computes the average precision, capturing how well
1674 the description separates activating from non-activating examples in a retrieval setting.

1675

1676 **clarity** Unlike the other metrics, clarity evaluates a feature description by testing whether
1677 it can generate activating text. Here, the evaluator model is prompted with the description and

1674 asked to generate candidate examples (full prompts are shown in Listings 11 and 12). We make 10
 1675 independent generation calls, producing a set of positives, and sample an equal number of random
 1676 examples as negatives. Each example is then passed through the subject model to obtain its activation
 1677 value. `clarity` is computed as the Gini index (a rescaled ROC AUC), measuring whether generated
 1678 examples achieve higher activations than the random negatives.

1679

1680

1681

1682

faithfulness `faithfulness` measures how well the feature description captures the feature's
 1683 causal role in the LLM. Unlike the other metrics that are based on analyzing activating data,
 1684 `faithfulness` compares the feature description against steered model generations where the fea-
 1685 ture is amplified or ablated. Here, we sample 10 random examples. Then we have the model
 1686 continue each example by generating 30 additional tokens. We apply this strategy multiple times,
 1687 where each time the feature is ablated (set to 0) or amplified. In the amplified settings, we set its
 1688 strength equal to its maximum known data activation multiplied by its modification factor (1, 10,
 1689 100). Given the steered generations and the feature description, we follow the same rating task as
 1690 in `responsiveness` and `purity`, asking the evaluator model to provide a discrete judgment (0, 1,
 1691 or 2) indicating how strongly the text matches the description. Following FADE, samples labeled
 1692 as 1 are discarded before scoring. The evaluator model prompts are shown in Listings 9 and 10.
 1693 We compute `faithfulness` as the maximum proportion of matching generations at each modification
 1694 factor compared to the proportion of matching generations when the feature is ablated.

1695

```

<system>
You are an intelligent and meticulous linguistics researcher.

You will be given a certain latent of text, such as "male pronouns" or "text with negative sentiment".

You will then be given several text examples. Your task is to determine which examples possess the latent.

For each example in turn, return 1 if the sentence is correctly labeled or 0 if the tokens are mislabeled.
You must return your response in a valid Python list. Do not return anything else besides a Python list.
</system>

<user>
Latent explanation: Words related to American football positions, specifically the tight end position.

1703 Test examples:
1704 Example 0:<|endoftext|>Getty Images\n\nPatriots tight end Rob Gronkowski had his boss'
1705 Example 1: names of months used in The Lord of the Rings:\n\n"...the
1706 Example 2: Media Day 2015\n\nLSU defensive end Isaiah Washington (94) speaks to the
1707 Example 3: shown, is generally not eligible for ads. For example, videos about recent tragedies,
1708 Example 4: line, with the left side - namely tackle Byron Bell at tackle and guard Amini
</user>

<assistant>
[1,0,1,0,1]
</assistant>

<user>
Latent explanation: The word 'guys' in the phrase 'you guys'.

1712 Test examples:
1713 Example 0: if you are<< comfortable>> with it. You<< guys>> support me in many other ways already and
1714 Example 1: birth control access<|endoftext|> but I assure you<< women>> in Kentucky aren't laughing as they
1715 struggle
1716 Example 2:>'s gig! I hope you guys<< LOVE>> her, and<< please>> be nice,
1717 Example 3:American, told<< Hannity>> that "you<< guys>> are playing the race card."
1718 Example 4:<< the>><|endoftext|>I want to<< remind>> you all that 10 days ago (director Massimil
</user>

<assistant>
[0,0,0,0,0]
</assistant>

<user>
Latent explanation: "of" before words that start with a capital letter.

1721 Test examples:
1722 Example 0: climate, Tomblin's Chief of Staff Charlie Lorensen said.\n
1723 Example 1: no wonderworking relics, no true Body and Blood of Christ, no true Baptism
1724 Example 2: Deborah Sathe, Head of Talent Development and Production at Film London,
1725 Example 3: It has been devised by Director of Public Prosecutions (DPP)
1726 Example 4: and fair investigation not even include the Director of Athletics? Finally, we believe the
</user>

<assistant>
[1,1,1,1,1]
</assistant>

```

1728

1729

1730 Listing 5: The evaluation model prompt and few-shot examples used to compute the detection
 1731 scores of natural language feature descriptions. <system></system>, <user></user>, and
 1732 <assistant></assistant> delineate the message roles passed to the evaluation model.

1733

1734

```

<system>
You are an intelligent and meticulous linguistics researcher.

You will be given a certain latent of text formatted as a Semantic Regex. {SEMANTIC_REGEX_DESCRIPTION}

You will then be given several text examples. Your task is to determine which examples possess the latent.

For each example in turn, return 1 if the sentence is correctly labeled or 0 if the tokens are mislabeled.
You must return your response in a valid Python list. Do not return anything else besides a Python list.

</system>

<user>
Semantic Regex explanation: [:field American football position:]

Test examples:
Example 0:<|endoftext|>Getty Images\n\nPatriots tight end Rob Gronkowski had his boss'
Example 1: names of months used in The Lord of the Rings:\n\n"...the
Example 2: Media Day 2015\n\nLSU defensive end Isaiah Washington (94) speaks to the
Example 3: shown, is generally not eligible for ads. For example, videos about recent tragedies,
Example 4: line, with the left side - namely tackle Byron Bell at tackle and guard Amini
</user>

<assistant>
[1,0,1,0,1]
</assistant>

<user>
Semantic Regex explanation: [:symbol you guys:]

Test examples:
Example 0: if you are<< comfortable>> with it. You<< guys>> support me in many other ways already and
Example 1: birth control access<|endoftext|> but I assure you<< women>> in Kentucky aren't laughing as they
struggle
Example 2:'s gig! I hope you guys<< LOVE>> her, and<< please>> be nice,
Example 3:American, told<< Hannity>> that "you<< guys>> are playing the race card."
Example 4:<< the>><|endoftext|>I want to<< remind>> you all that 10 days ago (director Massimil
</user>

<assistant>
[0,0,0,0,0]
</assistant>

<user>
Semantic Regex explanation: [:symbol of:] [:field Capitalized Word:]

Test examples:
Example 0: climate, Tomblin's Chief of Staff Charlie Lorensen said.\n
Example 1: no wonderworking relics, no true Body and Blood of Christ, no true Baptism
Example 2: Deborah Sathe, Head of Talent Development and Production at Film London,
Example 3: It has been devised by Director of Public Prosecutions (DPP)
Example 4: and fair investigation not even include the Director of Athletics? Finally, we believe the
</user>

<assistant>
[1,1,1,1,1]
</assistant>

```

1768

1769

1770

1771

1772

1773

1774

1775

1776

1777

1778

1779

1780

1781

Listing 6: The evaluation model prompt and few-shot examples used to compute the detection scores of semantic regex feature descriptions. <system></system>, <user></user>, and <assistant></assistant> delineate the message roles passed to the evaluation model. At prompt time, {SEMANTIC_REGEX_DESCRIPTION} is replaced with the semantic regex description in Listing 1.

```

<system>
You are an intelligent and meticulous linguistics researcher.

You will be given a certain latent of text, such as "male pronouns" or "text with negative sentiment".

You will be given a few examples of text that contain this latent. Portions of the sentence which strongly
represent this latent are between tokens << and >>.

Some examples might be mislabeled. Your task is to determine if every single token within << and >> is
correctly labeled. Consider that all provided examples could be correct, none of the examples could be
correct, or a mix. An example is only correct if every marked token is representative of the latent

For each example in turn, return 1 if the sentence is correctly labeled or 0 if the tokens are mislabeled.
You must return your response in a valid Python list. Do not return anything else besides a Python list.

```

```

1782 </system>
1783
1784 <user>
1785 Latent explanation: Words related to American football positions, specifically the tight end position.
1786 Test examples:
1787 Example 0:<|endoftext|>Getty Images\n\nPatriots<< tight end>> Rob Gronkowski had his boss'
1788 Example 1: posted<|endoftext|>You should know this<< about>> offensive line coaches: they are large,
1789 demanding<< men>>
1790 Example 2: Media Day 2015\n\nLSU<< defensive>> end Isaiah Washington (94) speaks<< to the>>
1791 Example 3:<< running backs>>," he said. ... Defensive<< end>> Carroll Phillips is improving and his injury
1792 is
1793 Example 4:<< line>>, with the left side - namely<< tackle>> Byron Bell at<< tackle>> and<< guard>> Amini
1794 </user>
1795 <assistant>
1796 [1,0,1,0,1]
1797 </assistant>
1798 <user>
1799 Latent explanation: The word 'guys' in the phrase 'you guys'.
1800 Test examples:
1801 Example 0: if you are<< comfortable>> with it. You<< guys>> support me in many other ways already and
1802 Example 1: birth control access<|endoftext|> but I assure you<< women>> in Kentucky aren't laughing as they
1803 struggle
1804 Example 2:'s gig! I hope you guys<< LOVE>> her, and<< please>> be nice,
1805 Example 3:American, told<< Hannity>> that "you<< guys>> are playing the race card."
1806 Example 4:<< the>><|endoftext|>I want to<< remind>> you all that 10 days ago (director Massimil
1807 </user>
1808 <assistant>
1809 [0,0,0,0,0]
1810 </assistant>
1811 <user>
1812 Latent explanation: "of" before words that start with a capital letter.
1813 Test examples:
1814 Example 0: climate, Tomblin's Chief Chief<< of>> Staff Charlie Lorensen said.\n
1815 Example 1: no wonderworking relics, no true Body and Blood<< of>> Christ, no true Baptism
1816 Example 2: Deborah Sathe, Head<< of>> Talent Development and Production at Film London,
1817 Example 3: It has been devised by Director<< of>> Public Prosecutions (DPP)
1818 Example 4: and fair investigation not even include the Director<< of>> Athletics? Finally, we believe the
1819 </user>
1820 <assistant>
1821 [1,1,1,1,1]
1822 </assistant>

```

Listing 7: The evaluation model prompt and few-shot examples used to compute the fuzzing scores of natural language feature descriptions. <system></system>, <user></user>, and <assistant></assistant> delineate the message roles passed to the evaluation model.

```

1823
1824 <system>
1825 ou are an intelligent and meticulous linguistics researcher.
1826
1827 You will be given a certain latent of text formatted as a Semantic Regex. {SEMANTIC_REGEX_DESCRIPTION}
1828
1829 You will be given a few examples of text that contain this latent. Portions of the sentence which strongly
1830 represent this latent are between tokens << and >>.
1831
1832 Some examples might be mislabeled. Your task is to determine if every single token within << and >> is
1833 correctly labeled. Consider that all provided examples could be correct, none of the examples could be
1834 correct, or a mix. An example is only correct if every marked token is representative of the latent
1835
1836 For each example in turn, return 1 if the sentence is correctly labeled or 0 if the tokens are mislabeled.
1837 You must return your response in a valid Python list. Do not return anything else besides a Python list.
1838
1839 </system>
1840
1841 <user>
1842 Semantic Regex explanation: [:field American football position:]
1843
1844 Test examples:
1845 Example 0:<|endoftext|>Getty Images\n\nPatriots<< tight end>> Rob Gronkowski had his boss'
1846 Example 1: posted<|endoftext|>You should know this<< about>> offensive line coaches: they are large,
1847 demanding<< men>>
1848 Example 2: Media Day 2015\n\nLSU<< defensive>> end Isaiah Washington (94) speaks<< to the>>
1849 Example 3:<< running backs>>," he said. ... Defensive<< end>> Carroll Phillips is improving and his injury
1850 is
1851 Example 4:<< line>>, with the left side - namely<< tackle>> Byron Bell at<< tackle>> and<< guard>> Amini
1852 </user>
1853 <assistant>
1854 [1,0,1,0,1]
1855 </assistant>

```

```

1836
1837 <user>
1838 Semantic Regex explanation: [:symbol you guys:]
1839
1840 Test examples:
1841 Example 0: if you are<< comfortable>> with it. You<< guys>> support me in many other ways already and
1842 Example 1: birth control access<|endoftext|> but I assure you<< women>> in Kentucky aren't laughing as they
1843 struggle
1844 Example 2:'s gig! I hope you guys<< LOVE>> her, and<< please>> be nice,
1845 Example 3:American, told<< Hannity>> that "you<< guys>> are playing the race card."
1846 Example 4:<< the>><|endoftext|>I want to<< remind>> you all that 10 days ago (director Massimil
1847 </user>
1848
1849 <assistant>
1850 [0,0,0,0,0]
1851 </assistant>
1852
1853 <user>
1854 Semantic Regex explanation: [:symbol of:] [:field Capitalized Word:]
1855
1856 Test examples:
1857 Example 0: climate, Tomblin's Chief Chief<< of>> Staff Charlie Lorensen said.\n
1858 Example 1: no wonderworking relics, no true Body and Blood<< of>> Christ, no true Baptism
1859 Example 2: Deborah Sathe, Head<< of>> Talent Development and Production at Film London,
1860 Example 3: It has been devised by Director<< of>> Public Prosecutions (DPP)
1861 Example 4: and fair investigation not even include the Director<< of>> Athletics? Finally, we believe the
1862 </user>
1863
1864 <assistant>
1865 [1,1,1,1,1]
1866 </assistant>

```

1867 Listing 8: The evaluation model prompt **and few-shot examples used** to compute the fuzzing
 1868 scores of semantic regex feature descriptions. **<system></system>, <user></user>, and**
 1869 **<assistant></assistant>** delineate the message roles passed to the evaluation model.
 1870 At prompt time, **{SEMANTIC_REGEX_DESCRIPTION}** is replaced with the semantic regex
 1871 description in Listing 1.

```

1868 <system>
1869 You are tasked with building a database of sequences that best represent a specific concept.
1870 To create this, you will review a dataset of varying sequences and rate each one according to how much the
1871 concept is expressed.
1872
1873 For each sequence, assign a rating based on this scale:
1874
1875 0: The concept is not expressed.
1876 1: The concept is vaguely or partially expressed.
1877 2: The concept is clearly and unambiguously present.
1878
1879 Use conservative ratings. If uncertain, choose a lower rating to avoid including irrelevant sequences in
1880 your database.
1881 If no sequence expresses the concept, rate all sequences as 0.
1882
1883 Each sequence is identified by a unique ID. Provide your ratings as a Python dictionary with sequence IDs
1884 as keys and their ratings as values.
1885
1886 Example Output: {"14": 0, "15": 2, "20": 1, "27": 0}
1887
1888 Output only the dictionary - no additional text, comments, or symbols.
1889 </system>

```

1890 Listing 9: The evaluation model prompt used **to compute** the responsiveness and purity scores
 1891 of natural language feature descriptions. **<system></system>** delineates the message roles
 1892 passed to the evaluation model.

```

1893 <system>
1894 You are tasked with building a database of sequences that best represent a specific concept.
1895 To create this, you will review a dataset of varying sequences and rate each one according to how much the
1896 concept is expressed.
1897
1898 The concept will be written as a Semantic Regex. {SEMANTIC_REGEX_DESCRIPTION}
1899
1900 For each sequence, assign a rating based on this scale:
1901
1902 0: The concept is not expressed.
1903 1: The concept is vaguely or partially expressed.
1904 2: The concept is clearly and unambiguously present.
1905
1906 Use conservative ratings. If uncertain, choose a lower rating to avoid including irrelevant sequences in
1907 your database.
1908 If no sequence expresses the concept, rate all sequences as 0.
1909
1910 Each sequence is identified by a unique ID. Provide your ratings as a Python dictionary with sequence IDs
1911 as keys and their ratings as values.

```

1890

```

1891 Example Output: {"14": 0, "15": 2, "20": 1, "27": 0}
1892 Output only the dictionary - no additional text, comments, or symbols.
1893 </system>

```

1894

Listing 10: The evaluation model prompt used to compute the responsiveness and purity scores of semantic regex feature descriptions. <system></system> delineates the message roles passed to the evaluation model. At prompt time, {SEMANTIC_REGEX_DESCRIPTION} is replaced with the semantic regex description in Listing 1.

1895

1896

1897

1898

1899

1900

```

<system>
You are tasked with building a database of sequences that best represent a specific concept.
To create this, you will generate sequences that vary in style, tone, context, length, and structure, while
maintaining a clear connection to the concept.
The concept does not need to be explicitly stated in each sequence, but each should relate meaningfully to
it. Be creative and explore different ways to express the concept.

Here are examples of how different concepts might be expressed:

Concept: "German language" - Sequences might include German phrases, or sentences.
Concept: "Start of a Java Function" - Sequences might include Java code snippets defining a function.
Concept: "Irony" - Sequences might include ironic statements or expressions.

Provide your sequences as strings in a Python List format.

Example: ["This is a first example sequence.", "Second example sequence but it is much longer also there
are somy typos in it. wjo told you that I can type?"]

Output only the Python List object, without any additional comments, symbols, or extraneous content.
</system>

```

1901

1902

1903

1904

1905

1906

1907

1908

1909

1910

1911

Listing 11: The evaluation model prompt used to compute the clarity scores of natural language feature descriptions. <system></system> delineates the message roles passed to the evaluation model.

1912

1913

1914

1915

1916

1917

1918

1919

1920

1921

1922

1923

1924

1925

1926

1927

1928

1929

```

<system>
You are tasked with building a database of sequences that best represent a specific concept.
To create this, you will generate sequences that vary in style, tone, context, length, and structure, while
maintaining a clear connection to the concept.

The concept will be expressed as a Semantic Regex. {SEMANTIC_REGEX_DESCRIPTION}
Be creative and explore different ways to express the concept, while faithfully expressing the semantic
regex.

Here are examples of how different concepts might be expressed:

Concept: "[topic German Language]" - Sequences might include German phrases, or sentences.
Concept: "@{[Java]}(functions)" - Sequences might include Java code snippets defining a function.
Concept: "[lexeme irony]" - Sequences that include the string 'irony', 'ironic', 'ironically', etc.

Provide your sequences as strings in a Python List format.

Example: ["This is a first example sequence.", "Second example sequence but it is much longer also there
are somy typos in it. wjo told you that I can type?"]

Output only the Python List object, without any additional comments, symbols, or extraneous content.
</system>

```

1930

1931

1932

1933

Listing 12: The evaluation model prompt used to compute the clarity scores of semantic regex feature descriptions. <system></system> delineates the message roles passed to the evaluation model. At prompt time, {SEMANTIC_REGEX_DESCRIPTION} is replaced with the semantic regex description in Listing 1.

1934

1935

1936

1937

1938

1939

1940

1941

1942

1943