WHAT DO BENCHMARKS MEASURE? DISCOVERING BENCHMARK DEMANDS AND GROUNDING SYSTEM CAPABILITIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern LLM benchmarks increasingly evaluate AI systems on realistic and complex scenarios, where individual items often involve multiple overlapping demands. However, the demand structure underlying these items remains hidden, making it difficult to determine what benchmarks actually measure. Existing fine-grained evaluation approaches typically rely on predefined or constructed evaluation dimensions, which may not capture the compositional nature of complex benchmark demands. We introduce BENCHSCOPE, a framework that discovers benchmark demands from item semantics and grounds their behavioral relevance through system responses. BENCHSCOPE first learns sparse and interpretable demand factors from benchmark item representations using a sparse autoencoder (SAE) without access to system responses. These fixed factors define a demand space for a multidimensional item response model (IRT). The model separates overall system performance, semantic item difficulty, and demand-specific capability specialization. Across seven benchmarks, BENCHSCOPE recovers reproducible demand factors and shows that they capture meaningful system performance differences beyond overall performance and item difficulty. The resulting benchmark and system profiles reveal that benchmarks targeting similar tasks can emphasize different demand compositions, while systems with similar aggregate scores can exhibit distinct capability specializations.

1 INTRODUCTION

Large language model benchmarks provide a standard framework for evaluating AI systems across diverse tasks (Hendrycks et al., 2021; Liang et al., 2023). As AI systems become more capable (Guo et al., 2025; Yang et al., 2025), benchmarks have evolved from simple task-oriented evaluations toward more realistic and complex scenarios (Jimenez et al., 2024; Wang et al., 2026b;a). In these complex benchmarks, individual items often involve multiple, overlapping demands rather than a single isolated demand. However, the demand composition underlying each benchmark item is often hidden, making it increasingly difficult to determine what benchmarks actually measure.

Understanding what benchmarks measure involves two related questions. First, what demands are present in a benchmark? Second, how strongly does each demand contribute to individual items? Existing approaches mainly address the first question by defining benchmark dimensions in advance. They typically rely on expert knowledge or heuristic rules to construct evaluation taxonomies. These taxonomies are then used to map benchmark items onto predefined dimensions (Zhang et al., 2026; Kuang et al., 2025; Maimon et al., 2026). Such approaches are useful when the relevant dimensions are known. However, for complex benchmarks, the appropriate demand decomposition is often unclear. Individual items may combine multiple demands, while predefined dimensions may merge distinct demands or overlook dimensions important for system performance. The second question, namely quantifying demand composition, has received much less attention. Addressing it requires first discovering the underlying demand structure and then estimating its composition, which is challenging when this structure is not known in advance.

We therefore ask whether benchmark demands can be discovered from item semantics and used to ground system capabilities, rather than being specified in advance. To address this question,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

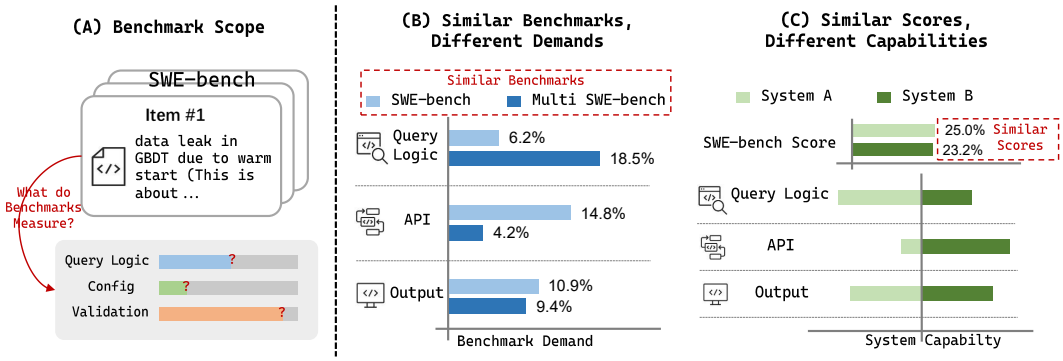


Figure 1: **Motivation.** Aggregate scores obscure the structure of benchmark demands and system capabilities. (A) The demand structure of a benchmark is not explicitly specified. (B) Similar benchmarks can emphasize different semantic demands. (C) Systems with similar aggregate scores can exhibit substantially different capability profiles.

we introduce BENCHSCOPE, a framework that first discovers demand structure from benchmark content and then grounds these demands through system behavior. Specifically, we train a Sparse AutoEncoder (SAE) on benchmark item representations without access to system responses, uncovering sparse and interpretable semantic demand factors. These factors define a fixed demand space. Using these fixed factors, we model system responses with a multidimensional Item Response Theory (IRT) model. The model separates overall system performance, semantic item difficulty, and demand-specific capability specialization, allowing behavioral evidence to validate the discovered demands without redefining the semantic axes.

The resulting framework provides a shared and interpretable basis for analyzing both benchmark demands and system capabilities. Across seven agent benchmarks, we recover reproducible and interpretable demand factors. We show that discovered demand factors capture meaningful differences in system performance beyond overall performance and item difficulty. Using these grounded demands, we construct benchmark and system profiles that characterize benchmark demand compositions and system capability specializations. These profiles reveal that benchmarks targeting similar tasks may emphasize different demand compositions, while systems with similar aggregate scores can exhibit distinct capability specializations.

Our contributions are summarized as follows:

- We formulate benchmark interpretation as the problem of discovering latent demand structures from benchmark items and grounding their behavioral relevance through system responses, without assuming predefined evaluation dimensions.
- We introduce BENCHSCOPE, a framework that separates semantic demand discovery from behavioral grounding. It discovers interpretable demand factors from item semantics and grounds their behavioral relevance through item response modeling.
- We systematically evaluate BENCHSCOPE across seven benchmarks, demonstrating the reproducibility, interpretability, and behavioral relevance of discovered demands. We further construct benchmark and system profiles to characterize benchmark demand compositions and system capability specializations.

2 RELATED WORK

Fine-Grained Evaluation of LLM Benchmarks. Fine-grained evaluation seeks to understand what benchmarks measure by identifying the demands underlying benchmark items and characterizing system performance across these demands. Existing approaches typically rely on predefined or constructed dimensions to characterize benchmark requirements. Zhang et al. (2026) define domain-specific ability taxonomies and item–ability associations to estimate capability-level performance, while Wang et al. (2026c) automatically construct hierarchical capability taxonomies from query-derived tags. Other approaches infer latent skill structures from behavioral evidence. Maimon et al.

(2026) recover latent skills from model performance patterns using factor analysis, whereas Kim et al. (2025) analyze benchmark composition through gradient-based importance and targeted parameter ablation over predefined cognitive abilities. However, existing approaches either rely on predefined dimensions or infer dimensions from behavioral patterns, making it difficult to characterize unknown demand structures and item-level demand composition. In contrast, BENCHSCOPE discovers continuous demand factors from benchmark-item semantics and grounds their behavioral relevance through system responses while keeping the semantic axes fixed.

Behavioral Modeling for LLM Evaluation. Item response theory (IRT) provides a principled framework for jointly modeling system ability and item characteristics (Reckase, 2006). Recent studies adapt IRT and related models to language-model evaluation for modeling item difficulty, system ability, and latent capability structure. TINYBENCHMARKS (Polo et al., 2024) estimates full-benchmark performance from small item subsets. IrtNet (Chen et al., 2025) models multi-dimensional abilities for performance prediction and routing. More closely related to capability analysis, Zhang et al. (2026) combine multidimensional IRT with explicit item–ability associations, while Liu et al. (2026) incorporate text embeddings to estimate item–attribute structures. However, these approaches typically require predefined or externally derived attributes to specify the dimensions used for behavioral modeling. In contrast, BENCHSCOPE anchors system–item interactions to demand factors discovered independently from system responses, keeping the semantic axes fixed during behavioral modeling. This enables testing the behavioral relevance of discovered demands while separating overall system performance, semantic item difficulty, and demand-specific capability specialization.

Semantic Representation Learning for Benchmark Analysis. Sparse autoencoders (SAEs) decompose dense representations into sparse and often interpretable features (Bricken et al., 2023). Recent studies investigate SAE scaling and feature evaluation, demonstrating their ability to recover interpretable representations from high-dimensional neural features (Gao et al., 2025). Beyond mechanistic interpretation, Jiang et al. (2026) apply SAEs to embedding spaces for dataset analysis, including dataset comparison, correlation discovery, clustering, and property-based retrieval. Building on this perspective, BENCHSCOPE uses SAEs to discover recurring demand factors from benchmark-item representations. Unlike approaches that treat semantic interpretability as sufficient evidence, we independently evaluate whether discovered factors are behaviorally relevant through system responses.

3 METHOD

3.1 PROBLEM SETUP AND BACKGROUND

Problem Setup. Consider a benchmark with N items evaluated by M systems. Let x_j denote the content of item j, and let $y_{mj} \in \{0, 1\}$ indicate whether system m successfully solves item j. The system responses form a matrix $Y \in \{0, 1\}^{M \times N}$.

We assume that a complex benchmark item may involve a mixture of multiple latent demands rather than belong to a single predefined category. Our goal is to discover these demand dimensions from item semantics and then determine whether they correspond to systematic differences in system behavior. Demand discovery is performed without access to Y, while system responses are used only for subsequent behavioral grounding.

Item Response Theory Model. Item response theory (IRT) models system–item outcomes using latent system ability and item difficulty (Reckase, 2006). In its basic ability–difficulty form,

$$P(y_{mj} = 1) = \sigma(g_m - b_j), \tag{1}$$

where g_m denotes the overall ability of system m, b_j denotes the difficulty of item j, and $\sigma(\cdot)$ is the logistic function. The parameters are estimated from observed system responses by maximizing the Bernoulli likelihood. This formulation captures overall system strength and item difficulty, but does not model how systems differ across the latent demands within an item.

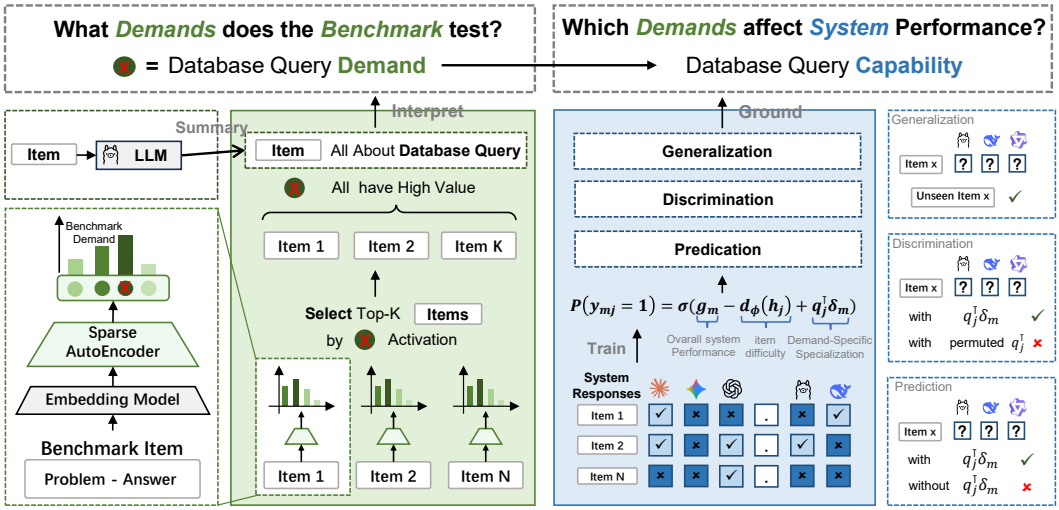


Figure 2: **Overview of BenchScope.** BenchScope first discovers semantic demand factors from benchmark item content and then grounds them in system behavior using a capability-anchored IRT model. The model separates overall system performance, item difficulty, and demand-specific specialization while keeping the semantic axes fixed.

3.2 Discovering Benchmark Semantic Demand Factors

We first encode each item using a frozen embedding model, obtaining $e_j = E(x_j)$. When an item contains multiple complementary views, such as an issue description, reference patch, and test patch, we encode each view separately and concatenate their embeddings to form the item representation.

Sparse AutoEncoder. To discover recurring semantic structure in benchmark items, we train a sparse autoencoder (SAE) on their embeddings:

$$h_j = \mathrm{Sparse}(W_e e_j + b_e), \qquad \hat{e}_j = W_d h_j + b_d, \tag{2}$$

where W_e and W_d are the encoder and decoder matrices, b_e and b_d are bias terms, $\mathrm{Sparse}(\cdot)$ denotes the sparsification operation, and $h_j \in \mathbb{R}^L$ is the sparse latent representation with width L. The SAE is trained to reconstruct input embeddings while enforcing sparse latent activations.

Stable Benchmark Demand Factors. To identify reproducible demand factors, we train the SAE with multiple random seeds and align factors across runs using Hungarian matching (Kuhn, 1955) based on their activation patterns over benchmark items. We retain factors with consistent activation patterns and overlapping top-activating items across runs, while filtering out rarely activated features. Let $\mathcal{S}$ denote the retained stable factors and $K = |\mathcal{S}|$. For each item, we define its normalized demand vector as $q_j = \mathrm{Normalize}(h_j[\mathcal{S}]) \in \mathbb{R}^K_{\geq 0}$, where nonzero retained activations are normalized to sum to one. Items with no retained activation are treated as uncovered.

Semantic Factor Interpretation. For each stable factor, we collect its top-activating items and use a large language model to summarize their shared semantic pattern into a concise factor description. To validate the description, a separate LLM annotator is given only the factor description and a disjoint set of benchmark items and asked to predict factor activation. We compare these predictions with the actual SAE activations to measure how well the description captures the learned factor. The descriptions are used only for interpretation and do not affect the learned factor axes or subsequent behavioral grounding.

3.3 Grounding Semantic Demands in System Capabilities

Semantic coherence alone does not establish behavioral relevance. We therefore ask whether system performance varies systematically along the discovered demand factors. The semantic axes are fixed

before behavioral modeling, so system responses are used only to test their relevance and cannot redefine the discovered structure.

Capability-Anchored Item Response Model. Building on the ability–difficulty formulation, we anchor system–item interactions to the discovered demand factors:

$$P(y_{mj} = 1) = \sigma\left(g_m - d_\phi(h_j) + q_j^\top \delta_m\right), \tag{3}$$

where g_m captures overall system performance, $d_\phi(h_j)$ predicts item difficulty from the full SAE representation using a two-layer MLP, and $q_j^\top \delta_m$ captures demand-specific system specialization. The difficulty model uses the full SAE representation, while system-specific interactions are restricted to stable demand factors, ensuring that specialization captures behavioral variation beyond item difficulty.

To separate overall performance from relative specialization, we impose $\bar{q}^\top \delta_m = 0$, where $\bar{q}$ is the mean demand composition of covered training items, and center the overall system effects across the evaluated population. Under this parameterization, g_m represents performance at the average demand composition, while $\delta_{mk} > 0$ and $\delta_{mk} < 0$ indicate relative strength and weakness along demand k, respectively.

For each system, we further define its capability representation as

$$\theta_m = g_m \mathbf{1} + \delta_m, \tag{4}$$

so that $g_m + q_j^\top \delta_m = q_j^\top \theta_m$ for covered items. Each entry θ_{mk} combines the system's overall performance level with its relative specialization along demand k, and is used later to construct the system capability profile.

Behavioral Grounding Tests. Estimating the capability parameters alone does not establish behavioral relevance. We therefore evaluate the discovered demands through three complementary tests: prediction, discrimination, and generalization.

(i) Prediction. We test whether demand-specific interactions improve held-out prediction beyond overall system performance and semantic item difficulty. We compare the full model with the ability–difficulty baseline

$$P^{(0)}(y_{mj} = 1) = \sigma\left(g_m - d_\phi(h_j)\right), \tag{5}$$

and compute $\Delta\text{LogLoss} = \mathcal{L}^{(1)}_{\text{test}} - \mathcal{L}^{(0)}_{\text{test}}$. A negative value indicates additional predictive signal from the discovered demand structure.

(ii) Discrimination. We test whether this signal depends on the discovered semantic axes rather than additional interaction parameters. We permute q_j across items and refit the same model, preserving model capacity while breaking semantic alignment. Better performance with the original assignments indicates that the gain depends on the discovered demand structure.

(iii) Generalization. We test whether specialization learned from training items generalizes to unseen items expressing the same demand. For held-out responses, we compute

$$r_{mj} = y_{mj} - \hat{P}^{(0)}(y_{mj} = 1), \qquad R_{mk} = \frac{\sum_{j \in \mathcal{H}_k} q_{jk} r_{mj}}{\sum_{j \in \mathcal{H}_k} q_{jk}}, \tag{6}$$

where $\mathcal{H}_k$ contains held-out items with strong activation on factor k. We compare $\hat{\delta}_{mk}$ with R_{mk} across systems using Spearman correlation and pairwise ranking agreement. Positive agreement indicates generalization to unseen demand-heavy items.

Together, these tests provide complementary evidence for whether discovered semantic demands are behaviorally grounded rather than merely interpretable.

3.4 Benchmark and System Profiles

The grounded demand space supports complementary benchmark and system profiles. Benchmark profiles characterize which demands are emphasized by the benchmark, while system profiles characterize system capabilities along the behaviorally grounded demand dimensions.

For each retained factor k, we define its benchmark demand mass and item coverage as

$$B_k = \frac{\sum_{j=1}^{N} q_{jk}}{\sum_{j=1}^{N}\sum_{\ell=1}^{K} q_{j\ell}}, \qquad C_k = \frac{1}{N}\sum_{j=1}^{N} \mathbb{I}[q_{jk} > 0], \tag{7}$$

where B_k measures the relative mass of demand k among the retained demands, and C_k measures the fraction of benchmark items activating that demand.

For each system m, we use θ_m on the behaviorally grounded factors as its capability profile. Each θ_{mk} represents the system's capability level along demand k, combining its overall performance with its relative specialization on that demand.

4 EXPERIMENTS

Our experiments evaluate BENCHSCOPE through three research questions that follow its discover–ground–profile pipeline. **RQ1: Semantic Discovery.** Are the discovered demand factors stable and interpretable without behavioral supervision? **RQ2: Behavioral Grounding.** Do discovered semantic demands correspond to behaviorally meaningful capability differences? **RQ3: Benchmark and System Profiles.** What do the resulting demand and capability profiles reveal about benchmark composition and system capabilities?

4.1 EXPERIMENTAL SETUP

Benchmarks. We evaluate BENCHSCOPE on seven agent benchmarks spanning software engineering, competitive programming, code optimization, API interaction, and terminal use: SWE-bench Verified (Jimenez et al., 2024), SWE-bench Pro (Deng et al., 2025), Multi-SWE-bench (Zan et al., 2026), LiveCodeBench (Jain et al., 2025), GSO (Shetty et al., 2026), AppWorld (Trivedi et al., 2024), and Terminal-Bench 2.0 (Merrill et al., 2026). For all benchmark, we randomly assign 80% of items to training and 20% to testing, using the same split for SAE training and IRT fitting so that test items are unseen by both.

Implementation Details. We encode benchmark items with the frozen Qwen3-Embedding-0.6B encoder and train a BatchTopK sparse autoencoder with latent width $L = 64$ and sparsity $k_{\text{act}} = 4$.The SAE is trained with seeds $\{0, 1, 42\}$, and factors across seeds are aligned using Hungarian matching (Kuhn, 1955) on pairwise activation similarity. We retain a factor as stable if its minimum pairwise activation correlation across seeds is at least 0.60, its mean Jaccard@20 overlap is at least 0.30, and its activation prevalence is at least 1%. Only these stable factors are used for subsequent demand-specific analysis. We use `deepseek-v4-flash` with temperature 0 to generate semantic descriptions from high-activating items and validate these descriptions on disjoint items. Further details on cross-seed SAE factor alignment, stable-factor selection criteria, and SAE hyperparameter selection are provided in the Appendix.

Evaluation Details. For semantic discovery, we evaluate cross-seed stability, benchmark coverage, and semantic validity of factor descriptions. Stability is measured by factor consistency across SAE runs, while semantic validity is evaluated by a LLM annotator predicting factor activation from descriptions on disjoint items, measured by macro balanced accuracy and AUROC. For behavioral grounding, we evaluate prediction through held-out log loss, discrimination through matched permutation controls, and generalization through cross-system Spearman correlation and pairwise ranking agreement. We additionally evaluate factor-specific generalization using cross-system Spearman correlation and pairwise ranking agreement, and use matched permutation controls to test whether behavioral gains depend on the discovered semantic axes. Missing observations are excluded from evaluation.

4.2 RQ1: SEMANTIC DISCOVERY

Semantic discovery recovers reproducible demand factors. Across benchmarks, the SAE recovers 11–38 stable demand factors out of 64, covering 55.1–99.2% of benchmark items (Table 1). SWE-bench Verified yields the largest stable set, with 38 factors covering 99.2% of items,

Benchmark	Stable/M	Coverage	Dec. Cos.	BAcc.	AUROC
SWE-bench Verified	38/64	99.2%	0.812	0.725	0.903
Multi-SWE-bench	28/64	80.8%	0.817	0.746	0.875
SWE-bench Pro	21/64	81.8%	0.715	0.747	0.861
LiveCodeBench	12/64	66.3%	0.868	0.750	0.844
AppWorld	16/64	68.7%	0.619	0.831	0.908
GSO	21/64	82.0%	0.415	0.662	0.757
Terminal-Bench 2.0	11/64	55.1%	0.426	0.732	0.686

Table 1: **Stability and semantic validity of discovered factors.** Stable/M, Coverage, and Dec. Cos. measure cross-seed stability and coverage, while BAcc. and AUROC evaluate the semantic validity of factor descriptions.

Benchmark	ΔLL $\downarrow$	Perm. $p\downarrow$	Mean $\rho\uparrow$	PairAcc. $\uparrow$
SWE-bench Verified	-0.000938^*	0.0099	0.182	0.562
Multi-SWE-bench	-0.001259^*	0.0099	0.129	0.548
SWE-bench Pro	-0.000352	0.0297	0.200	0.566
LiveCodeBench	-0.000383^*	0.0099	0.150	0.553
AppWorld	-0.003635^*	0.0099	0.596	0.729
GSO	-0.000107^*	0.0099	0.178	0.582
Terminal-Bench 2.0	-0.000527^*	0.0099	0.169	0.551

Table 2: **Behavioral grounding across benchmarks.** ΔLL measures predictive improvement over the ability–difficulty baseline (*: 95% bootstrap CI excludes zero). Perm. p evaluates whether predictive gains depend on the alignment between discovered demands and items, while Mean ρ and PairAcc. measure specialization generalization to unseen items.

while Multi-SWE-bench and SWE-bench Pro also retain substantial stable structure. Decoder cosine similarity shows strong cross-seed agreement on several benchmarks, reaching 0.868 on LiveCodeBench, although agreement is weaker for GSO and Terminal-Bench 2.0, where smaller benchmark sizes make stable factor estimation more challenging.

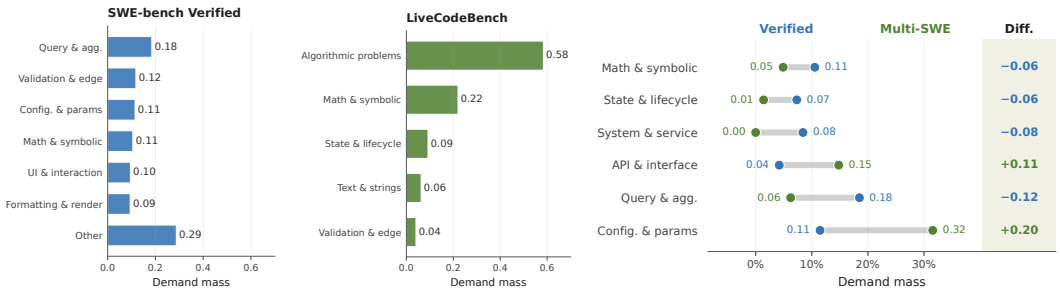
Stable factors admit interpretable semantic descriptions. We next evaluate whether the recovered factors correspond to semantic patterns that generalize beyond the items used for description. Given only a factor description and the content of a disjoint evaluation item, an LLM annotator predicts whether the factor is active. Comparing these predictions with SAE activations yields macro balanced accuracy of 0.662–0.831 and AUROC of 0.686–0.908 (Table 1). These results indicate that the generated descriptions capture recurring semantic patterns underlying factor activations rather than merely summarizing individual high-activating examples.

Overall, RQ1 demonstrates that semantic discovery can recover reproducible and interpretable demand factors without behavioral supervision.

4.3 RQ2: Behavioral Grounding

RQ1 shows that the discovered demand factors are reproducible and semantically interpretable. We next ask whether these fixed semantic demands are behaviorally relevant, explaining systematic differences in system performance beyond overall ability and item difficulty. We evaluate this through three tests: prediction, discrimination, and generalization.

Prediction: Demand-specific interactions improve held-out prediction. We first test whether demand-specific interactions improve held-out prediction beyond the ability–difficulty baseline. As shown in Table 2, the capability-anchored model achieves lower held-out log loss on all seven benchmarks. The bootstrap confidence intervals exclude zero for six benchmarks, with the largest improvement on AppWorld, followed by Multi-SWE-bench and SWE-bench Verified. SWE-bench Pro also shows a negative point estimate, although its confidence interval includes zero, indicating weaker evidence of improvement. Overall, these results provide evidence that the discovered demand factors are behaviorally relevant beyond overall ability and item difficulty.



(a) SWE-bench Verified (b) LiveCodeBench (c) SWE-bench Verified vs. Multi-SWE-bench

Figure 3: **Benchmark demand profiles.** Semantically related discovered factors are grouped for visualization. (a), (b) SWE-bench Verified and LiveCodeBench exhibit distinct demand compositions. (c) Benchmarks within the same broad domain can still emphasize different demands.

Discrimination: Predictive signal depends on semantic alignment. We next test whether the predictive signal depends on the correspondence between discovered demands and benchmark items, rather than additional interaction parameters alone. We permute demand vectors among difficulty-matched items and refit the same interaction model for 100 draws. As shown in Table 2, the observed improvement is more extreme than the permutation null on six benchmarks ($p = 0.0099$), while SWE-bench Pro also shows significant separation from the null ($p = 0.0297$). These results indicate that predictive signal arises from semantic alignment between items and discovered demands rather than interaction capacity alone.

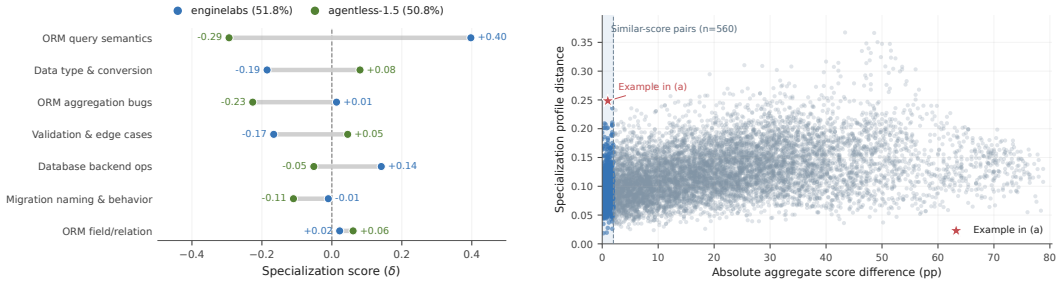
Generalization: Demand-specific specialization generalizes to unseen items. We finally test whether demand-specific specialization learned from training responses generalizes to unseen items expressing the same demand. We estimate system specialization on training items and evaluate whether the resulting specialization ordering is preserved on held-out items after residualizing the ability–difficulty baseline. As shown in Table 2, all seven benchmarks show positive agreement: mean Spearman ρ ranges from 0.129 to 0.596, and PairAcc. is above 0.5 throughout. AppWorld shows the strongest association ($\rho = 0.596$, PairAcc. $= 0.729$). Overall, the learned demand-specific specialization generalizes to unseen items with the same semantic demand.

Overall, RQ2 provides evidence that the discovered semantic demands correspond to behaviorally meaningful capability differences beyond overall ability and item difficulty. This grounding enables the construction of benchmark and system capability profiles in RQ3.

4.4 RQ3: Benchmark and System Profiles

RQ1 shows that BenchScope discovers stable and interpretable semantic demands, while RQ2 establishes their behavioral relevance. We finally investigate how the grounded demand space characterizes benchmark composition and system capability specialization beyond coarse benchmark labels and aggregate scores.

Benchmark profiles reveal distinct demand compositions. We characterize each benchmark by aggregating the demand mass of its discovered factors. For cross-benchmark visualization, we manually group semantically related factors into shared demand categories based on their descriptions. These categories serve only as an interpretable summary of the discovered factors; all quantitative analyses are conducted on the original factor space. Figure 3 shows that SWE-bench Verified and LiveCodeBench exhibit distinct demand compositions. SWE-bench Verified contains larger demand proportions for factors associated with software-engineering operations, such as query and aggregation logic, data validation, configuration handling, and interface behavior, whereas LiveCodeBench contains more factors related to algorithmic problem solving and mathematical reasoning. Similar differences also exist among benchmarks within the same broad domain. For example, after summarizing related factors into shared categories, SWE-bench Verified and Multi-SWE-bench exhibit different demand compositions, with Multi-SWE-bench placing relatively more emphasis on configuration handling and API/interface consistency. Overall, these results show that benchmark



(a) Similar scores, different specialization

(b) Score difference vs. specialization distance

Figure 4: **System capability specialization beyond aggregate scores.** (a) Two SWE-bench Verified systems with nearly identical aggregate accuracy exhibit substantially different capability specialization profiles. (b) Across system pairs, substantial specialization-profile differences remain even when aggregate score differences are small.

demand profiles provide a finer-grained characterization of benchmark demands beyond coarse task or domain labels.

System profiles reveal differences hidden by aggregate scores. We next examine whether systems with similar aggregate performance can exhibit different capability specialization patterns. We first identify pairs of systems with closely matched benchmark scores and compare their grounded specialization profiles. As shown in Figure 4(a), two SWE-bench Verified systems with nearly identical accuracy nevertheless show clear differences across several grounded demands. We then extend this analysis to all system pairs by comparing aggregate score differences with specialization-profile distance. Figure 4(b) shows that substantial profile differences remain even among systems with very similar scores. Overall, these results show that aggregate benchmark scores can hide meaningful differences in how systems specialize across behaviorally grounded demands.

Overall, RQ3 shows that the grounded demand space provides complementary views of benchmark composition and system capability specialization. It characterizes the demands emphasized by benchmarks and the capability differences among systems beyond coarse benchmark labels and aggregate scores.

5 Conclusion

We introduced BenchScope, a framework for discovering benchmark demands from item semantics and grounding their behavioral relevance without assuming a predefined capability taxonomy. BenchScope first discovers recurring semantic demand factors from benchmark items alone and then anchors system–item interactions to these fixed factors to evaluate their behavioral relevance. Across seven agent benchmarks, we show that the discovered factors are reproducible and semantically interpretable, while the grounded demand space captures system-specific variation beyond overall ability and item difficulty and generalizes to unseen items expressing the same demands. The resulting benchmark and system profiles provide a fine-grained view of benchmark compositions and system capability specializations beyond coarse labels and aggregate scores.

BenchScope also has several limitations. First, the discovered demand factors depend on the semantic representations provided by the embedding model and the capacity of the sparse autoencoder, and different representation choices may reveal complementary demand structures. Second, behavioral grounding requires system response data, which limits its direct application to newly constructed benchmarks or systems without sufficient evaluations. Future work may explore more adaptive demand discovery methods, broader behavioral evidence sources, and continual analysis of evolving benchmarks and systems.

AI USE STATEMENT

In this work, we used generative AI tools to assist with the writing and editing of the manuscript, including improving readability, refining the presentation of research ideas, and providing feedback on the organization of the methodology and experiments. We also used generative AI tools to assist with the creation and refinement of scientific figures.

We have not used generative AI tools to generate datasets, develop the core research methodology, formulate mathematical claims, produce experimental results, or replace the authors' analysis and interpretation of the findings. These tasks were performed by the authors.

Additionally, we used generative AI tools for recommended disclosure tasks, including editing the manuscript for clarity, suggesting improvements to paper structure, assisting with figure design, and supporting literature summarization. All AI-assisted content was reviewed and verified by the authors. We checked the correctness of AI-assisted descriptions, experimental interpretations, and generated artifacts through manual verification and comparison with the original analyses. We take responsibility for the final content of this work, including all text, claims, and artifacts produced with the aid of generative AI tools.

ETHICS STATEMENT

This work analyzes existing benchmark datasets and publicly available system evaluation results. It does not involve human subjects, private user data, or the collection of new personal information. The proposed framework is intended for analyzing and improving the understanding of AI system evaluations. We acknowledge that benchmark-based analyses may inherit limitations from the underlying datasets and evaluation protocols, and we encourage careful interpretation of capability-related conclusions.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of BENCHSCOPE and its evaluation protocols to facilitate reproducibility. The main paper describes the overall framework, including semantic demand discovery, behavioral grounding, and benchmark/system profile construction. The experimental setup, benchmark configurations, model settings, evaluation metrics, and validation procedures are described in the Experiments section, while additional implementation details, hyperparameter selection, factor stability analysis, and ablation studies are provided in the Appendix.

REFERENCES

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

Jianhao Chen, Chenxu Wang, Gengrui Zhang, Peng Ye, Lei Bai, Wei Hu, Yuzhong Qu, and Shuyue Hu. Learning compact representations of llm abilities via item response theory, 2025. URL https://arxiv.org/abs/2510.00844.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa Kundurthy, Sean Hendryx, Zifan Wang, Vijay Bharadwaj, Jeff Holm, Raja Aluri, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks?, 2025. URL https://arxiv.org/abs/2509.16941.

Leo Gao, Tom Dupre la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. Scaling and evaluating sparse autoencoders. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=tcsZt9ZNKD.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

Nicholas Jiang, Xiaoqing Sun, Lisa Dunlap, Lewis Smith, and Neel Nanda. Interpretable embeddings with sparse autoencoders: A data analysis toolkit. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=E98OvY1Syv.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Dongjun Kim, Gyuho Shim, Yongchan Chun, Minhyuk Kim, Chanjun Park, and Heuiseok Lim. Benchmark profiling: Mechanistic diagnosis of LLM benchmarks. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 15635–15650, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.789. URL https://aclanthology.org/2025.emnlp-main.789/.

Ziyan Kuang, Feiyu Zhu, Maowei Jiang, Yanzhao Lai, Zelin Wang, Zhitong Wang, Meikang Qiu, Jiajia Huang, Min Peng, Qianqian Xie, and Sophia Ananiadou. From scores to skills: A cognitive diagnosis framework for evaluating financial large language models, 2025. URL https://arxiv.org/abs/2508.13491.

Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Cosgrove, Christopher D Manning, Christopher Re, Diana Acosta-Navas, Drew A. Hudson, Eric Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue WANG, Keshav Santhanam, Laurel Orr, Lucia Zheng, Mert Yuksekgonul, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri S. Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Andrew Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. Holistic evaluation of language models. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=iO4LZibEqW. Featured Certification, Expert Certification, Outstanding Certification.

Jia Liu, Zhiyu Xu, and Yuqi Gu. Scalable text-embedding-informed cognitive diagnosis of large language models, 2026. URL https://arxiv.org/abs/2603.14676.

Aviya Maimon, Amir David Nisan Cohen, Gal Vishne, Shauli Ravfogel, and Reut Tsarfaty. From benchmarks to skills: Low-rank factors for llm evaluation. *Transactions of the Association for Computational Linguistics*, 14:1639–1653, 07 2026. ISSN 2307-387X. doi: 10.1162/TACL.a.745. URL https://doi.org/10.1162/TACL.a.745.

Mike A Merrill, Alexander Glenn Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Kumar Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Kwesi Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Jenia Jitsev, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=a7Qa4CcHak.

Felipe Maia Polo, Lucas Weber, Leshem Choshen, Yuekai Sun, Gongjun Xu, and Mikhail Yurochkin. tinybenchmarks: evaluating LLMs with fewer examples. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=qAml3FpfhG.

Mark D Reckase. 18 multidimensional item response theory. *Handbook of statistics*, 26:607–642, 2006.

Manish Shetty, Naman Jain, Jinjian Liu, Vijay Kethanaboyina, Koushik Sen, and Ion Stoica. GSO: Challenging software optimization tasks for evaluating SWE-agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2026. URL https://openreview.net/forum?id=I5qDL315bQ.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *ACL*, 2024.

Zhun Wang, Nico Schiller, Hongwei Li, Srijiith Sesha Narayana, Milad Nasr, Nicholas Carlini, Xiangyu Qi, Eric Wallace, Elie Bursztein, Luca Invernizzi, Kurt Thomas, Yan Shoshitaishvili, Wenbo Guo, Jingxuan He, Thorsten Holz, and Dawn Song. Exploitgym: Can ai agents turn security vulnerabilities into real attacks? *arXiv preprint arXiv:2605.11086*, 2026a.

Zhun Wang, Tianneng Shi, Jingxuan He, Matthew Cai, Jialin Zhang, and Dawn Song. Cybergym: Evaluating AI agents' real-world cybersecurity capabilities at scale. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=2YvbLQEdYt.

Zongqi Wang, Tianle Gu, Chen Gong, Xin Tian, Siqi Bao, and Yujiu Yang. SCAN: Structured capability assessment and navigation for LLMs. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2760–2799, San Diego, California, United States, July 2026c. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.126. URL https://aclanthology.org/2026.acl-long.126/.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, et al. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Shulin Xin, Linhao Zhang, Qi Liu, Aoyan Li, Lu Chen, Xiaojian Zhong, Siyao Liu, Yongsheng Xiao, Liangqiang Chen, Yuyu Zhang, Jing Su, Tianyu Liu, RUI LONG, Ming Ding, and liang xiang. Multi-SWE-bench: A multilingual benchmark for issue resolving. In *The Thirty-ninth Annual Conference on Neural Information*

Processing Systems Datasets and Benchmarks Track, 2026. URL https://openreview.net/forum?id=MhBZzkz4h9.

Xu Zhang, Xudong Gong, Jiacheng Qin, Qiang Wang, JiaQi Liao, Zhe Wang, Dawei Feng, and Bo Ding. Beyond scores: Diagnostic llm evaluation via fine-grained abilities, 2026. URL https://arxiv.org/abs/2604.12191.

Benchmark	Tasks	Systems	Obs.	Density	Success
SWE-bench Verified	500	134	67,000	1.00	51.5%
Multi-SWE-bench	2,132	83	78,486	0.44	10.9%
SWE-bench Pro	730	14	10,220	1.00	25.8%
LiveCodeBench	1,055	73	68,015	0.88	64.8%
Terminal-Bench 2.0	89	112	9,968	1.00	42.8%
AppWorld	585	14	8,190	1.00	15.6%
GSO	102	15	1,530	1.00	12.5%

Table 3: **Benchmark statistics.** Density denotes the fraction of observed entries in each system–item response matrix. Success is computed over the observed binary outcomes used in our experiments.

A BENCHMARK DETAILS

We evaluate BENCHSCOPE on seven benchmarks covering repository-level software engineering, competitive programming, software optimization, interactive API use, and terminal-based agent tasks. Table 3 reports the exact subsets used in our experiments. These counts correspond to the intersection of available benchmark content and public system-level evaluation records after preprocessing, and therefore need not equal the nominal size of the original benchmark. Missing system–item outcomes are treated as missing observations rather than failures.

SWE-bench Verified. SWE-bench Verified is a human-validated subset of SWE-bench that evaluates agents on real-world GitHub issues requiring repository modifications and successful test execution (Jimenez et al., 2024). We use the 500-item SWE-bench Verified benchmark in our experiments. We collect public results from 134 agent systems, forming a binary response matrix, where an item is marked successful if the corresponding system resolves the issue. SWE-bench Verified provides issue descriptions, reference patches, and verification patches, which are used to construct semantic representations of benchmark items.

Multi-SWE-bench. Multi-SWE-bench extends repository-level issue resolution to multiple programming languages (Zan et al., 2026). We use 2,132 tasks from the public `verified` evaluations across C, C++, Go, Java, JavaScript, Python, Rust, and TypeScript, including the 500-item Python subset corresponding to SWE-bench Verified, together with results from 83 systems. Since systems are evaluated on different subsets of languages, the resulting response matrix contains 78,486 observed entries with density 0.44; missing entries are treated as unobserved rather than failures. Binary outcomes are derived from the released `resolved` sets, while task representations use the pull-request title and body together with the available reference patch and metadata.

SWE-bench Pro. SWE-bench Pro evaluates agents on more challenging, long-horizon repository-level software-engineering tasks (Deng et al., 2025). Its public split contains 731 instances, of which 730 have behavioral results in our data; we use these 730 tasks with 14 systems, yielding a complete matrix of 10,220 binary responses. Semantic representations are constructed from the official problem statements, reference patches, test patches, and repository metadata.

LiveCodeBench. LiveCodeBench evaluates code-generation systems on continuously updated competitive-programming problems from platforms such as LeetCode, AtCoder, and Codeforces (Jain et al., 2025). Our data contain 1,055 distinct problems and 73 systems, with 68,015 observed system–item outcomes (density 0.88) because different systems were evaluated on different benchmark releases. Each item is assigned a binary success if any generation in its released `graded_list` passes the tests; therefore, these outcomes are used as analysis labels rather than as the official Pass@1 metric. We represent each item using its problem statement, title, starter code, and available metadata.

Terminal-Bench 2.0. Terminal-Bench 2.0 evaluates agents on heterogeneous tasks performed through a sandboxed terminal environment, including software development, debugging, environment manipulation, and related command-line workflows (Merrill et al., 2026). We use all 89 tasks

Embedding	NMSE ↓	Stable/M ↑	Dec. Cos. ↑	Pearson ↑	J@20 ↑	Coverage ↑
Qwen3-0.6B	0.6495	**33/64**	0.7545	0.8339	0.5238	**96.8%**
BGE-M3	0.7691	26/64	0.6878	0.8078	0.5253	96.3%
Jina-v2-Code	**0.5444**	26/64	0.7545	**0.8616**	**0.6001**	92.5%

Table 4: **Embedding-model ablation.** All models use a BatchTopK SAE with $M = 64$, $K = 4$, and I+P+V input views. Stable/M denotes the fraction of cross-seed stable factors, and J@20 denotes top-20 activating-item overlap across matched factors.

together with public binary outcomes from 112 systems, giving a complete matrix of 9,968 observations. The released task descriptions, tests, categories, tags, and difficulty metadata are used to construct semantic representations.

AppWorld. AppWorld evaluates interactive agents on multi-step tasks carried out through APIs in a simulated ecosystem of everyday applications (Trivedi et al., 2024). We combine the public `test_normal` and `test_challenge` splits, containing 168 and 417 non-overlapping tasks, respectively, and merge their results for 14 systems to obtain a complete 14×585 response matrix. Binary outcomes are taken from the official task-level `success` field, while the natural-language task instruction serves as the semantic input; unlike repository-level benchmarks, AppWorld provides no reference or verification patch.

GSO. GSO evaluates software-engineering agents on optimizing the runtime performance of real code while preserving correctness (Shetty et al., 2026). We use all 102 tasks and binary outcomes from 15 systems, yielding a complete response matrix of 1,530 observations. Since GSO tasks do not follow the conventional issue-repair format, we construct the semantic task description from the target API and performance-test script, while retaining the expert optimization diff and correctness tests as additional item information.

Response Processing. All benchmarks are represented as binary system–item response matrices. Operational failures such as missing generations, unapplied patches, timeouts, environment failures, or incorrect solutions are mapped to $y_{mj} = 0$ whenever the corresponding benchmark record marks the task as unsuccessful, while missing system–item evaluations remain unobserved and are excluded from the IRT likelihood. All evaluated items have corresponding content for semantic representation, although the available fields differ across benchmarks; for example, AppWorld and LiveCodeBench do not provide reference and verification patches in the same form as repository-level software-engineering benchmarks. BENCHSCOPE therefore constructs item representations from the semantic information available for each benchmark rather than requiring a uniform set of input views.

B SAE CONFIGURATION ABLATIONS

We select the semantic representation and SAE configuration on the 1,794 SWE-bench test instances outside SWE-bench Verified. The discovery corpus is stratified by repository into 1,435 training and 359 validation instances using a fixed split shared across all configurations, and each setting is trained with seeds $\{0, 1, 42\}$. We evaluate configurations by reconstruction quality, sparsity, coverage, and cross-run reproducibility. Factors are aligned across seeds with Hungarian matching and considered stable when they achieve a minimum pairwise activation Pearson correlation of 0.60, mean Jaccard@20 of 0.30, and minimum activation prevalence of 0.01 on the validation split. We prioritize stable-factor recovery and cross-seed agreement, using reconstruction and coverage as complementary selection criteria.

B.1 EMBEDDING MODEL

We first compare embedding models while fixing BatchTopK, $M = 64$, $K = 4$. Table 4 reports the results.

M	K	NMSE ↓	Dead ↓	Stable/M ↑	Dec. Cos. ↑	Pearson ↑	J@20 ↑	Coverage ↑
32	1	0.8155	4.2%	6/32	0.7006	0.8334	0.6336	25.3%
32	2	0.7314	0.0%	16/32	**0.8319**	**0.8540**	0.5329	83.2%
32	4	0.7055	0.0%	14/32	0.6529	0.8101	0.5331	94.7%
64	1	0.9413	38.0%	8/64	0.2743	0.8310	0.6386	18.6%
64	2	0.9096	9.9%	18/64	0.2426	0.8246	0.5978	54.4%
64	**4**	**0.6495**	**0.0%**	**33/64**	**0.7545**	**0.8339**	**0.5238**	**96.8%**
64	8	0.6248	0.0%	29/64	0.5368	0.8041	0.5141	99.5%
128	1	0.9673	57.0%	5/128	0.1850	0.7626	0.5365	18.8%
128	2	0.9477	31.0%	11/128	0.1422	0.7787	0.5964	38.3%
128	4	0.9130	7.0%	27/128	0.1206	0.7664	0.5470	70.9%
128	8	0.5744	0.0%	51/128	0.5740	0.8005	0.4780	99.7%
128	16	0.5534	0.0%	25/128	0.3551	0.7862	0.4978	98.3%
256	1	0.9758	73.4%	2/256	0.1380	0.6180	0.0631	6.3%
256	2	0.9635	53.8%	9/256	0.1033	0.7932	0.6632	16.1%
256	4	0.9315	29.2%	15/256	0.0928	0.7726	0.5745	39.6%
256	8	0.8906	7.9%	25/256	0.0928	0.7542	0.4871	62.6%
256	16	0.5218	0.0%	43/256	0.3690	0.7628	0.4294	99.1%
512	1	0.9801	86.5%	3/512	0.1474	0.8349	0.6999	6.5%
512	2	0.9638	74.2%	12/512	0.1090	0.7874	0.6453	25.6%
512	4	0.9510	53.6%	8/512	0.0688	0.7814	0.5888	24.7%
512	8	0.9125	25.6%	20/512	0.0632	0.7776	0.5730	49.9%
512	16	0.8631	3.5%	36/512	0.0580	0.7259	0.4843	73.1%

Table 5: **Dictionary-width and sparsity ablation.** All configurations use Qwen3-Embedding-0.6B, BatchTopK, and I+P+V views. Stable/M denotes the number of cross-seed stable factors relative to dictionary width. Large dictionaries with small K frequently exhibit optimization failure, reflected by high dead-feature rates and very low decoder agreement.

Qwen3-Embedding-0.6B yields the largest stable-factor fraction, recovering 33 of 64 factors (51.6%), compared with 26 of 64 (40.6%) for both alternatives, while also achieving the highest validation coverage (96.8%). Jina-v2-Code obtains lower NMSE and stronger activation correlation and top-item overlap, and is therefore not uniformly dominated. Under our selection rule, which prioritizes reproducible factor recovery and coverage rather than reconstruction alone, we select Qwen3-Embedding-0.6B.

B.2 Dictionary Width and Sparsity

We next fix Qwen3-Embedding-0.6B and BatchTopK and vary dictionary width $M \in \{32, 64, 128, 256, 512\}$ and target sparsity $K \in \{1, 2, 4, 8, 16\}$, retaining combinations satisfying $K \leq M/8$. Table 5 reports all 22 valid configurations.

The sweep separates several degenerate settings from configurations with reliable cross-seed recovery: 14 of 22 settings exhibit either high dead-feature rates or mean decoder cosine below 0.30, often together with substantially earlier stopping. Among the remaining configurations, $(M, K) = (64, 4)$ and $(32, 2)$ achieve the highest stable-factor fractions (51.6% and 50.0%), while $(64, 4)$ provides better validation coverage (96.8% vs. 83.2%) and reconstruction (NMSE 0.6495 vs. 0.7314). Larger or denser settings can further improve reconstruction and coverage, but generally reduce stable-factor recovery or cross-seed agreement. We therefore select $M = 64$ and $K = 4$ as the best trade-off between reconstruction, coverage, sparsity, and reproducibility.

B.3 SAE Variant

Finally, we compare Vanilla L1, TopK, and BatchTopK SAEs using the selected Qwen3 representation, $M = 64$.

Vanilla L1 exhibits the strongest raw cross-seed stability, but its realized $L_0 = 9.77$ is almost twice that of BatchTopK and substantially denser than the intended sparse item decomposition. We there-

Variant	NMSE ↓	L0 ↓	Stable/M ↑	Pearson ↑	J@20 ↑	Coverage ↑
Vanilla L1	0.6117	9.77	**39/64**	**0.9023**	**0.5766**	**100.0%**
TopK	0.6225	**4.00**	31/64	0.8363	0.5129	93.0%
BatchTopK	**0.6089**	5.26	34/64	0.8419	0.5440	98.3%

Table 6: **SAE-variant ablation.** Comparison of reconstruction, sparsity, cross-seed stability, and validation coverage across Vanilla L1, TopK, and BatchTopK SAEs. All variants use Qwen3-Embedding-0.6B with $M = 64$.

Component	Frozen setting
Embedding	Qwen3-Embedding-0.6B
Dictionary width	$M = 64$
Target sparsity	$K = 4$
SAE variant	BatchTopK
Random seeds	$\{0, 1, 42\}$

Table 7: **Frozen SAE configuration used in the formal experiments.** All configuration choices are made exclusively on the non-Verified semantic-discovery corpus, without access to SWE-bench Verified items or system responses.

fore do not interpret the Vanilla comparison as a sparsity-matched win. The direct comparison between TopK and BatchTopK is better controlled because both use $K = 4$. BatchTopK improves NMSE (0.6089 vs. 0.6225), stable-factor recovery (34 vs. 31 factors), activation correlation (0.8419 vs. 0.8363), Jaccard@20 (0.5440 vs. 0.5129), and coverage (98.3% vs. 93.0%), while TopK obtains higher decoder cosine (0.7898 vs. 0.7603). We therefore retain BatchTopK.

B.4 Final Frozen Configuration

The ablations result in the following configuration, which is frozen before any evaluation on SWE-bench Verified or other benchmarks:

Under this configuration, the semantic validation split yields an NMSE of 0.6089, no dead features, 34/64 stable factors (53.1%), mean decoder cosine of 0.7603, mean activation Pearson correlation of 0.8419, mean Jaccard@20 of 0.5440, and 98.3% stable-factor coverage.