OPTIMAL SKILL SELECTION FOR LLM AGENTS WITH PROVABLE BICRITERIA GUARANTEES

Anonymous authors
Paper under double-blind review

ABSTRACT

Loading reusable skill documents into a bounded context window has become a primary way large language model (LLM) agents acquire task-specific capabilities, which makes skill selection a first-order determinant of task performance and token cost. Yet current agents score skills independently by semantic relevance and assemble the set by top-k or greedy packing, with no quality guarantee or cost awareness on the selected set. Redundant or poorly chosen skills then waste scarce context tokens and can even degrade performance. In this paper, we present a theory-grounded and practical framework for budgeted skill selection. We give the first model of how skill sets shape execution outcomes, capturing complementary capability coverage and diminishing returns from redundancy through a monotone submodular benefit, while accounting for context degradation with a linear token penalty under a hard budget. Based on this model, we develop Best Prefix Selection (`BPS`), a polynomial-time algorithm, and prove, to our knowledge, the first performance guarantee for skill selection: a bicriteria $(1 - 1/e, 1)$ approximation whose benefit coefficient is optimal in polynomial time. We construct a controlled testbed based on BigCodeBench to isolate the effect of skill selection on execution success. On it, `BPS` with a learned capability encoder reaches a success rate of 0.65, and the strongest baselines need at least 28% more tokens to reach 0.60.

1 INTRODUCTION

Large language model (LLM) agents increasingly rely on reusable skill documents to acquire task-specific capabilities beyond their parametric knowledge (Yang et al., 2026), and public skill registries already list tens of thousands of installable skills (Cho et al., 2026; Gao et al., 2026). Modern production agents such as Codex (OpenAI, 2026) and Claude Code (Anthropic, 2025) deploy skills through a two-stage mechanism of *selection* and *execution*: the LLM reviews each installed skill's metadata (name and description) and selects skills according to the query, then loads only the selected skill documents into its context window to solve the task.

However, both stages are limited by the model's finite context window: the context cost of selection scales with the size of the installed skill library, whereas execution consumes additional context for the selected skill documents and task input. Deployments are therefore confined to small libraries, and Codex, for instance, allots skill metadata at most 2% of the context window (OpenAI, 2026). Moreover, redundant content in the context disperses the model's attention and degrades execution (Liu et al., 2024; Modarressi et al., 2025), and even curated skills push success below the no-skill baseline on 13 of 87 benchmark tasks (Li et al., 2026b).

These limitations have motivated dedicated skill-selection mechanisms, from per-skill retrieval and routing to set-aware packing and context construction (Gan & Sun, 2025; Fore et al., 2024; Zheng et al., 2026b;a; Li et al., 2026c). Without a principled formulation, however, they remain heuristic. Skills are ranked by semantic relevance, a learned preference, or graph proximity, and the set is cut by top-k (Qu et al., 2024; Li et al., 2026a), truncation (Liu et al., 2026), or greedy packing (Zheng et al., 2026a). The resulting set carries no guarantee, and both its context cost and its capability overlap and complementarity are handled ad hoc.

Yet these capability relationships are critical to effective skill selection. Fig. 1 illustrates a common skill-selection paradigm for coding agents (OpenAI, 2026; Anthropic, 2025): given a task, the system selects skills from a library and provides them to a frozen LLM executor. We evaluate Qwen3-32B

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.

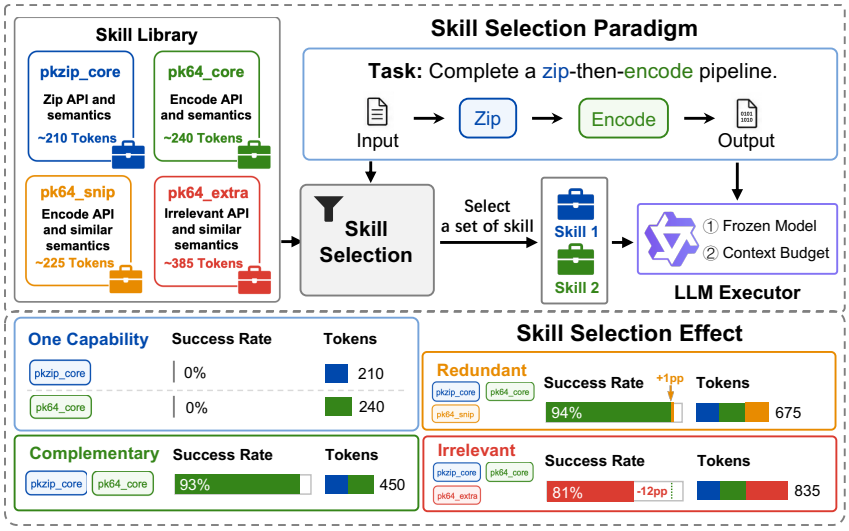

Figure 1: **Top:** skill selection in coding agents: a selector picks skills from a library and injects them into a frozen LLM executor. **Bottom:** selection is set-level: sets covering the required capabilities succeed, while redundant and irrelevant skills spend context budget for little or negative return.

(Yang et al., 2025) in this paradigm on tasks whose private APIs are accessible only through the selected skills, so execution depends directly on selection quality. We find that skills covering only one capability achieve zero success, whereas complementary skills covering both reach a 93% success rate. After both capabilities are covered, adding a redundant `pk64_snip` skill consumes 225 additional tokens but improves the success rate by only 1 percentage point, while adding a semantically related but task-irrelevant `pk64_extra` skill reduces it by 12 percentage points. These results establish skill selection as a budgeted set-level decision rather than a per-skill ranking.

Motivated by these observations, we formalize skill selection as a budgeted set decision within the selection–execution workflow of LLM agents (Section 3). We use a structured value model that can be fitted directly from execution outcomes while keeping the executor fixed. Each skill supplies latent capabilities and the query demands some of them; a concave response within each capability dimension captures diminishing returns from redundant coverage, while separate dimensions reward complementary coverage, so the gross benefit $G(S)$ is monotone submodular. Injected tokens are charged a linear context penalty $\kappa\ell(S)$ under a budget B:

$$\max_{S:\ell(S)\le B} F(S) := G(S) - \kappa\ell(S), \tag{1}$$

an instance of regularized submodular maximization under a knapsack constraint.

Solving this optimization with a provable guarantee, however, faces fundamental barriers. Without the penalty, the problem contains monotone submodular knapsack maximization, which admits no approximation better than $1 - 1/e$ unless P = NP (Feige, 1998). With the penalty, the objective may be negative, ruling out constant multiplicative approximation and motivating a bicriteria guarantee (Nikolakaki et al., 2021; Harshaw et al., 2019). Existing results for regularized submodular maximization under a hard budget obtain weaker approximation coefficients (Gong et al., 2024; Guo et al., 2026; Zhang & Luo, 2026) or incur an additive precision-dependent loss (Perrault et al., 2021).

We develop Best Prefix Selection (`BPS`, Algorithm 1), a practical polynomial-time algorithm for budgeted skill selection, which can be deployed as the selection stage in most popular LLM agent systems. We prove that `BPS` achieves a $(1 - 1/e, 1)$ bicriteria guarantee for the fitted objective in Eq. (7), with the best possible benefit coefficient and the full context penalty. To our knowledge, this work provides the first structured model of how the selected skill set shapes execution outcomes and the first skill-selection algorithm with a provable performance guarantee. On a controlled testbed with real LLM executions, the `BPS` pipeline outperforms all evaluated skill-selection baselines in average task success, while using fewer tokens than the strongest competitors. We summarize our main contributions as follows.

- **New Formulation.** We formalize skill selection as a regularized submodular maximization problem under a token budget constraint in Section 3. The structured objective makes redundancy, complementarity, and context cost explicit, and its parameters can be fitted from execution outcomes. We validate the model on real executions in Section 5.2.
- **Optimal Bicriteria Guarantee.** We develop a polynomial-time algorithm for the skill selection problem (`BPS`, Algorithm 1), and prove a $(1 - 1/e, 1)$ bicriteria guarantee for `BPS` with the best possible benefit coefficient (Theorem 1), using budget-aligned interpolation (Appendix A). In experiments, `BPS` reaches the best value found on 98% of selection instances with up to 124 skills, compared with 56% for the best baseline (Appendix B.6).
- **Experiments on Real Executions.** We construct a controlled testbed based on BigCodeBench (Zhuo et al., 2025) to isolate the effect of skill selection on execution success. On it, our structured objective Eq. (7) predicts execution success on unseen skill sets and recovers their hidden capability coverage. On tasks held out from training its neural capability encoder, the sets `BPS` selects reach 0.65 measured task success, above the 0.60 that the executor's own selection and a retriever fine-tuned on the same feedback reach on 28% and 36% more tokens, and above every released router and retriever at 0.22 or less (Sections 5.2 and 5.3).

2 Related Work

Skill selection for LLM agents. Skill retrieval and routing (Cho et al., 2026; Zheng et al., 2026b; Su et al., 2026; Xiao et al., 2026; Wang et al., 2026) narrow a large skill library to a candidate pool for downstream selection, following a pipeline inherited from tool use (Qin et al., 2024; Qu et al., 2024). SkillsInjector (Li et al., 2026c) learns how many skills to inject and renders them jointly; SkillSelect-Serve (Zheng et al., 2026a) greedily packs itemwise scores under token and deployment constraints; Graph-of-Skills (Liu et al., 2026) expands dependency-aware bundles under a context cap yet leaves its budgeted objective unsolved; GoSkills (Zeng et al., 2026) and SkillComposer (Zhao et al., 2026) assemble bounded skill groups and autoregressive subsets. To our knowledge, none of these heuristics states a provable guarantee for the injected set. Related methods provide guarantees for different selection problems: PACMS (Ghulyani et al., 2026) applies facility-location coverage to accumulated session content under a token knapsack, and the knapsack composer of (Yuan et al., 2025) admits agentic components online with a competitive ratio. In contrast, we formulate skill selection for a fixed executor as regularized submodular maximization under a hard token budget, explicitly modeling set-level redundancy and complementarity.

Regularized submodular maximization. Constrained submodular maximization is a classical template for placing content and services on capacity-limited resources (Golrezaei et al., 2012; Poularakis et al., 2019). For the pure benefit, density greedy with seed enumeration attains the tight $1 - 1/e$ (Sviridenko, 2004; Kulik et al., 2021; Feldman et al., 2023). Subtracting a linear penalty changes the problem's character: the objective can be negative, which rules out any constant multiplicative approximation (Nikolakaki et al., 2021), and distorted greedy (Harshaw et al., 2019) recovers a $(1 - 1/e, 1)$ bicriteria guarantee only under a cardinality constraint. Under a knapsack constraint, existing guarantees for the regularized objective include $(1/4, 1)$ (Gong et al., 2024), $((1 - 1/e)/2, 1/2)$ (Guo et al., 2026), and $(1/8 - \epsilon, 1)$ in one streaming pass (Zhang & Luo, 2026). Other results incur an additive $\kappa\epsilon$ loss with $O(B/\epsilon)$ budget levels (Perrault et al., 2021) or return a fractional solution (Feldman, 2021). ParetoGreedy (Vombatkere & Terzi, 2026) records all greedy prefixes as `BPS` does, but its instance-dependent Pareto analysis does not establish a uniform $(1 - 1/e, 1)$ guarantee for $G - \kappa\ell$. Our result establishes a uniform $(1 - 1/e, 1)$ guarantee for an integral solution without additive loss.

3 Model and Optimization Problem

We formulate skill selection as maximizing an improvement in task score under a context budget. Let $\mathcal{L} = \{s_1, \cdots, s_L\}$ denote the skill library, where s_i is the i-th skill document. Given a query q, the selection stage chooses a subset of skills, indexed by $S \subseteq [L]$, for a downstream executor E, held fixed throughout. Injecting the documents of the selected skills into the context window of E, let $J_E(q, S) \in \mathbb{R}$ be the score of the execution result on task q, with higher scores indicating better performance. The score can be an expected task grade, an expected human rating, or a utility of

success probability. We define the true score improvement over execution without skills as

$$F_E^\star(q, S) := J_E(q, S) - J_E(q, \varnothing). \tag{2}$$

The selection stage would ideally maximize $F_E^\star(q, S)$, but $F_E^\star$ is a black box. We therefore approximate it with a structured objective (Section 3.1), pose selection as maximizing it under a hard token budget (Section 3.2), and fit it from execution records, with empirical evidence supporting its use for skill selection (Section 3.3).

3.1 Key Observations and Structured Objective

The structured model rests on three observations. **(i)** Context value is query-dependent and set-level: a skill's contribution is not an intrinsic per-skill score but depends on whether its capabilities match those the query requires (Didolkar et al., 2024; An et al., 2023; Xu et al., 2024) and complement those the selected set already supplies (Gupta et al., 2023). **(ii)** More context is not uniformly beneficial: irrelevant content distracts execution (Shi et al., 2023), redundant items add little coverage (Gupta et al., 2023; Kumari et al., 2024), and longer inputs can hurt even when the relevant evidence is present (Liu et al., 2024). **(iii)** For a fixed executor and agent framework, the selected documents must fit the residual context budget (Ye et al., 2023; Kumari et al., 2024; Qu et al., 2024).

Structured Capability Benefit. Following observations (i) and (ii), we model the positive contribution of skills through a latent capability space with d dimensions. For each skill $s_i \in \mathcal{L}$, we assume there exists a latent capability supply vector $\boldsymbol{u}_i = (u_{i,1}, \cdots, u_{i,d}) \in \mathbb{R}_+^d$, where $u_{i,k}$ quantifies how much capability skill s_i supplies in the k-th dimension, e.g., operating git or analyzing logs.

To model complementarity and redundancy, a nondecreasing concave function h_k with $h_k(0) = 0$ turns the total supply into the coverage $h_k\left(\sum_{i\in S} u_{i,k}\right)$ of S in dimension k, so that supplies in the same dimension overlap redundantly while those in different dimensions remain complementary.

A latent capability demand vector $\boldsymbol{w}^q = (w_1^q, \ldots, w_d^q) \in \mathbb{R}_+^d$ weights the query's demand for each capability dimension. We model the gross score benefit as a monotone submodular function

$$G_E(q, S) := \sum_{k=1}^{d} \eta_k^E w_k^q \cdot h_k\left(\lambda_k^E \sum_{i\in S} u_{i,k}\right), \tag{3}$$

where $\eta_k^E \geq 0$ and $\lambda_k^E \geq 0$ are executor-specific parameters that calibrate E's sensitivity to the k-th capability dimension. This is the weighted-coverage form used in document summarization and data selection (Lin & Bilmes, 2011; Kirchhoff & Bilmes, 2014), with capability dimensions as features.

Degradation Penalty and Hard Context Budget. Following observation (ii), injected documents impose a cost that grows with context length. Let $\ell_i > 0$ be the token length of skill document s_i once injected, and $\ell(S) := \sum_{i\in S} \ell_i$ the total injected length. We approximate the resulting score degradation by $c_E(S) := \kappa_E \ell(S)$, where $\kappa_E \geq 0$ is the frozen executor's per-token sensitivity in the chosen score units. Length-sensitive execution motivates this first-order approximation: compressing skill-document bodies improves execution quality (Gao et al., 2026), and focused skills outperform larger, exhaustive ones (Li et al., 2026b).

Moreover, observation (iii) imposes a hard feasibility limit on skill selection. We take the budget B to be the residual token budget available to skill documents in the agent framework. The hard budget defines the feasible family $\mathcal{F}_B = \{S \subseteq [L] : \ell(S) \leq B\}$.

3.2 Optimization Problem in Skill Selection

Given the structured model, we formalize the selection stage as a constrained optimization problem. The selection objective combines the capability benefit with the degradation penalty,

$$F_E(q, S) := G_E(q, S) - c_E(S), \tag{4}$$

and the selection problem maximizes it over the feasible family:

$$\max_{S\in\mathcal{F}_B} F_E(q, S). \tag{5}$$

Problem Eq. (5) is computationally hard: setting $\kappa_E = 0$ and specializing the responses h_k to truncated sums recovers budgeted maximum coverage (Khuller et al., 1999), an NP-hard slice of

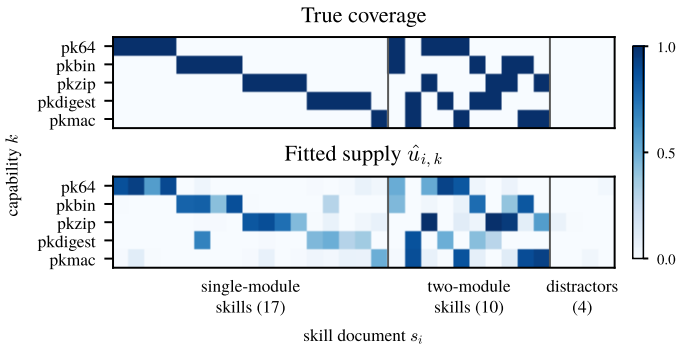

Figure 2: Parameter recovery. Columns are the 31 skill documents, grouped by the private modules they document: one, two, or none (distractors, which document look-alike APIs the tasks never use). Rows are the five capabilities, one per private module. Top: the true coverage from human labels. Bottom: the supply $\widehat{u}_{i,k}$ fitted from pass/fail outcomes alone, scaled by its largest entry, with latent dimensions matched to capabilities by the best permutation. Both panels share the 0–1 scale.

monotone submodular knapsack maximization (Sviridenko, 2004). The penalty can also make F_E non-monotone and negative, so guarantees for monotone submodular maximization do not apply directly. Section 4 develops an algorithm with a bicriteria guarantee that treats the benefit and the penalty asymmetrically.

3.3 Model Fitting and Empirical Validity

The structured objective Eq. (4) is built on latent quantities. Among its primitives, only the token length ℓ_i for each skill document s_i is observable. The capability supplies $\boldsymbol{u}_i$, the query demand $\boldsymbol{w}^q$, and the executor calibration parameters η_k^E, λ_k^E, κ_E admit no direct measurement. Two questions therefore decide whether the model is usable in practice: whether these latent quantities can be estimated from data, and whether an accurately estimated objective leads to a well-chosen skill set.

For the first question, the key observation is that the selection objective never requires the latent factors individually. In Eq. (3), the demand w_k^q enters only through the product $\eta_k^E w_k^q$, and the supply $u_{i,k}$ only through $\lambda_k^E u_{i,k}$. Defining the *effective demand* and *effective supply*

$$\widetilde{w}_k^q := \eta_k^E w_k^q, \quad \widetilde{u}_{i,k} := \lambda_k^E u_{i,k}, \quad \forall k \in [d], \tag{6}$$

the gross benefit Eq. (3) rewrites exactly as $G_E(q,S) = \sum_{k=1}^d \widetilde{w}_k^q \, h_k\big(\sum_{i\in S} \widetilde{u}_{i,k}\big)$. We therefore estimate the effective quantities directly with two *capability encoders*: a demand encoder $\widehat{\boldsymbol{\psi}}_E : q \mapsto \widehat{\boldsymbol{w}}^q$ and a supply encoder $\widehat{\boldsymbol{\phi}}_E : s_i \mapsto \widehat{\boldsymbol{u}}_i$, where $\widehat{\boldsymbol{w}}^q$ and $\widehat{\boldsymbol{u}}_i$ are estimates of the effective demand $\widetilde{\boldsymbol{w}}^q$ and effective supply $\widetilde{\boldsymbol{u}}_i$, respectively. Each encoder is instantiated as a text encoder with a nonnegative output layer, so that the fitted objective inherits the monotone submodularity established in Section 3.1. The context sensitivity is calibrated as $\widehat{\kappa}_E$ jointly with the encoders on execution records. Then we define the fitted objective

$$\widehat{F}_E(q,S) := \left\langle \widehat{\boldsymbol{\psi}}_E(q), \boldsymbol{h}\left(\sum_{i\in S} \widehat{\boldsymbol{\phi}}_E(s_i)\right)\right\rangle - \widehat{\kappa}_E \cdot \ell(S), \tag{7}$$

where $\boldsymbol{h}(\cdot)$ applies $\{h_k\}$ componentwise. At decision time, the selection layer solves the *fitted selection problem*, which is the deployable counterpart of Eq. (5) and is defined by

$$\max_{S\in\mathcal{F}_B} \widehat{F}_E(q,S). \tag{8}$$

Solving Eq. (8) provides a practical substitute for Eq. (5), with the latent quantities estimated from execution records. Its usefulness depends on whether Eq. (7) captures how skill combinations affect the frozen executor's performance. We assess this on the testbed of Section 5.1, examining its predictions, the capability structure it learns, and the execution outcomes of the sets it selects.

Algorithm 1 Best Prefix Selection (BPS)

Require: task q, executor E, skill library $\mathcal{L}$, budget B, fitted benefit oracle $\widehat{G}_E(q,\cdot)$, fitted context sensitivity $\widehat{\kappa}_E$.
Ensure: selected skill set $S_{\texttt{BPS}}$.
1: Discard every skill i with $\ell_i > B$; initialize the *prefix pool* $\mathcal{P} \leftarrow \emptyset$.
2: **for** each seed $A \subseteq [L]$ with $|A| \le 2$ and $\ell(A) \le B$ **do**
3: $S \leftarrow A$
4: Add prefix S to $\mathcal{P}$. *// each seed opens a prefix chain*
5: **while** some $i \notin S$ fits, i.e., $\ell(S) + \ell_i \le B$ **do**
6: $i^\star \leftarrow \operatorname{argmax}_{i \notin S,\, \ell(S)+\ell_i \le B} \left(\widehat{G}_E(q, \{i\} \cup S) - \widehat{G}_E(q,S)\right)/\ell_i$
7: $S \leftarrow S \cup \{i^\star\}$.
8: Add prefix S to $\mathcal{P}$. *// record every prefix in chain*
9: **end while**
10: **end for**
11: $S_{\texttt{BPS}} \leftarrow \operatorname{argmax}_{S \in \mathcal{P}} \widehat{F}_E(q,S)$. *// choose the best prefix*
12: **return** $S_{\texttt{BPS}}$

We detail the empirical evaluation in Section 5.2. Specifically, we fit Eq. (7) to pass/fail outcomes and test its predicted success probabilities. With minimal parameters, our model achieves the highest accuracy among the compared models. On the top panel of Figure 2, we label the 31 skills in the testbed by their coverage of 5 different capabilities. The bottom panel shows the supply $\widehat{u}_{i,k}$ fitted on all 5,204 measured task–set pairs of the testbed (Appendix B.2), with latent dimensions matched to capabilities by the best permutation. The fitted supplies recover the manually annotated skill-capability coverage with an AUC of 0.996. In Section 5.3 we also validate that optimizing the modeled objective improves actual execution.

4 Skill Selection with Provable Guarantees

We now present the selection algorithm and its per-instance guarantee for the fitted selection problem Eq. (8). For a fixed query q, frozen executor E, and budget B, the fitted objective in Eq. (7) is $\widehat{F}_E(q,S) = \widehat{G}_E(q,S) - \widehat{\kappa}_E \ell(S)$, where $\widehat{G}_E(q,S)$ is the fitted capability benefit and $\widehat{\kappa}_E$ is the calibrated context sensitivity. Our analysis uses only that $\widehat{G}_E(q,\cdot)$ is normalized ($\widehat{G}_E(q,\varnothing) = 0$), nondecreasing, and submodular, that all lengths ℓ_i are positive, and that $\widehat{\kappa}_E \ge 0$.

The objective $\widehat{F}_E(q,S) = \widehat{G}_E(q,S) - \widehat{\kappa}_E \ell(S)$ raises three difficulties at once. First, although $\widehat{G}_E(q,\cdot)$ is monotone, $\widehat{F}_E(q,\cdot)$ can be non-monotone and negative, so a selection rule must be allowed to stop early or output the empty set. In particular, the classical greedy analysis for monotone submodular maximization no longer applies to $\widehat{F}_E(q,\cdot)$. Second, skills have heterogeneous lengths under a single knapsack constraint, so locally dense choices are not enough. Third, the executor accepts only integral skill sets, so fractional reasoning must eventually land on an integral candidate. These difficulties call for a new analysis of approximate greedy algorithms.

We present our algorithm, *Best Prefix Selection* (BPS, Algorithm 1), a partial-enumeration density-greedy procedure with seed size two. In Lines 2–4, the algorithm enumerates all feasible seeds of size at most two. Then it grows a density-greedy chain from each seed by iteratively adding the skill with the highest marginal benefit per token (Lines 5–9), and records every feasible prefix encountered along each chain. Finally, it returns the single *best* recorded prefix, the one maximizing the fitted objective $\widehat{F}_E(q,\cdot)$ (Line 11).

Algorithm 1 implements the standard partial-enumeration density greedy for monotone submodular knapsack (Khuller et al., 1999; Sviridenko, 2004; Kulik et al., 2021). Under a monotone benefit, the endpoint of each chain dominates all its prefixes, so the known $1 - 1/e$ analyses compare only chain endpoints. Under the non-monotone fitted objective $\widehat{F}_E(q,\cdot)$, however, the endpoint need not be the best candidate, since the optimum can sit strictly inside a chain. Inspired by (Vombatkere & Terzi, 2026), we record every prefix and select the best one over the full collection by $\widehat{F}_E(q,\cdot)$; this best-prefix selection step gives BPS its name.

The theoretical guarantee for BPS is given below, with full proof in Appendix A.

Theorem 1 (Bicriteria $(1 - 1/e, 1)$-approximation guarantee)**.** *Let $\widehat{G}_E(q, \cdot)$ be normalized ($\widehat{G}_E(q, \varnothing) = 0$), nondecreasing, and submodular, let $\ell_i > 0$ for all $i \in [L]$, and let $\widehat{\kappa}_E \geq 0$. The BPS output S_{BPS} of Algorithm 1 satisfies $S_{BPS} \in \mathcal{F}_B$ and*

$$\widehat{F}_E(q, S_{BPS}) \geq \left(1 - \frac{1}{e}\right) \widehat{G}_E(q, T) - \widehat{\kappa}_E \ell(T) \quad \forall T \in \mathcal{F}_B. \tag{9}$$

Eq. (9) gives BPS a bicriteria guarantee for $\widehat{F}_E$, our learned approximation to the black-box objective $F^\star_E$, with coefficients $1 - 1/e$ on benefit and 1 on penalty. It controls optimization of the fitted objective but does not bound model error or, by itself, performance on $F^\star_E$. We evaluate the model's selection quality in Section 5.2 and demonstrate improved execution success with fewer tokens for the resulting BPS pipeline on our controlled testbed in Section 5.3.

Tightness. The benefit coefficient $\alpha = 1 - 1/e$ in Eq. (9) cannot be improved under standard complexity assumptions (any $(1 - 1/e + \epsilon)$-approximation is NP-hard (Feige, 1998)). Theorem 1 therefore provides a polynomial-time bicriteria guarantee with the best possible benefit coefficient for maximizing a monotone submodular benefit minus a linear penalty under a knapsack constraint.

Comparison to prior work. The guarantees closest to our setting address the same regularized objective under the same knapsack constraint, and each concedes what Theorem 1 does not: weaker approximation coefficients (Gong et al., 2024; Guo et al., 2026; Zhang & Luo, 2026), an additive precision-dependent loss (Perrault et al., 2021), or a fractional output (Feldman, 2021). ParetoGreedy (Vombatkere & Terzi, 2026) generates the same candidates as BPS, but its analysis gives instance-dependent Pareto bounds rather than a uniform $(1 - 1/e, 1)$ guarantee for the regularized objective. In contrast, Theorem 1 attains the $(1 - 1/e, 1)$ guarantee directly for an integral solution, with no additive loss.

Time Complexity. Algorithm 1 runs in $O(dL^4)$ time, where d is the capability dimension which can be selected a priori and L the library size: $O(L^2)$ seeds each grow a chain of at most L density steps, and each step scans $O(L)$ candidates at $O(d)$ cost per marginal evaluation. In practice L is the size of the shortlist left by a high-recall retrieval stage (Cho et al., 2026; Zheng et al., 2026b; Gan & Sun, 2025), not of the whole registry.

Empirical optimization quality. We also evaluate BPS on 900 skill-selection instances with libraries of 31 to 124 skills tiled from an objective fitted to real executor outcomes in Section 5.2. BPS reaches the best value found on 98% of them and outperforms all baselines, including Gong et al. (2024), whose share falls to 15% at 124 skills, and Guo et al. (2026) at 39% (Appendix B.6).

5 Evaluation

In this section, we evaluate the effectiveness of our proposed structured value model and the BPS algorithm. Details of the experimental setup and implementation are provided in Appendix B. We first introduce a controlled testbed in Section 5.1. Using this testbed, we evaluate the validity of the structured objective (Eq. (7)) in Section 5.2. We further propose an end-to-end skill selection protocol based on our methods in Section 5.3.

5.1 Controlled Testbed for Skill Selection

On existing benchmarks, success cannot be attributed entirely to the selected skills, because the executor may already solve the task from pretraining alone. Qwen3-32B (Yang et al., 2025) solves the original BigCodeBench (Zhuo et al., 2025) versions of our tasks in 85% of rollouts without any skill. To isolate the effect of skill selection, we build a testbed based on BigCodeBench by replacing standard library APIs with private variants (Appendix B.2).

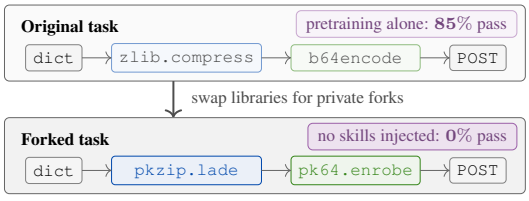

Figure 3: Forking a BigCodeBench task: standard libraries are replaced by private forks.

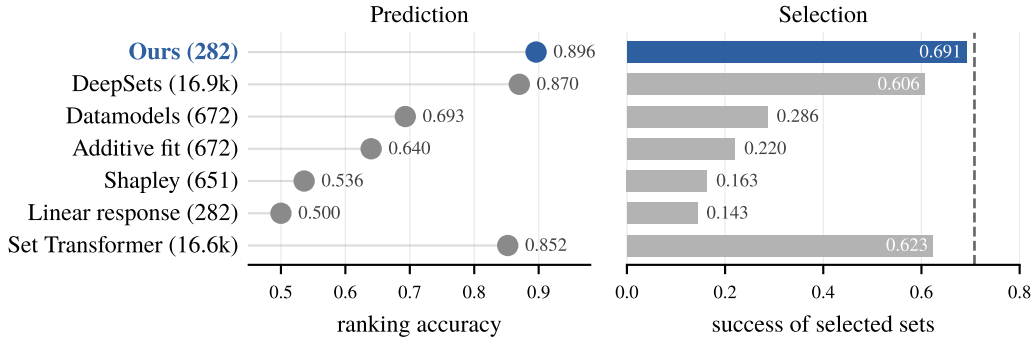

Figure 4: Value-model comparison, parameter counts in parentheses. *Left:* pairwise ranking accuracy on the held-out sets of the test tasks. *Right:* mean success rate of the selected sets on executions held out from fitting, with the empirical ceiling dashed.

As shown in Fig. 3, we fork BigCodeBench and replace its standard libraries with five private modules, one per capability family. The constructed skill library consists of skills documenting a single module or a pair of modules, as well as similar-looking but unrelated distractors. We exclude tasks that can be solved without using the constructed skills. On this testbed, the frozen executor Qwen3-32B records no successes in 100 executions without injected skills, which allows us to cleanly study the effect of skill selection on downstream task success and token cost. For each task and selected skill set used below, we run Qwen3-32B as the frozen executor and compute success rates from these real execution outcomes, more than 90,000 runs in total.

5.2 Validity of the Structured Objective

We evaluate whether higher scores from our structured value model in Eq. (7) correspond to higher execution success rates for the same task. The model is denoted by **Ours** in Figure 4. For this experiment, we learn a demand vector for each task and a supply vector for each skill in a five-dimensional capability space. The model has 282 learnable parameters. We detail the parameterization in Appendix B.3 and its connection to execution success and fitting procedure in Appendix B.4.

We compare our structured value model with several interpretable baselines. Additive fit and Shapley scores (Ghorbani & Zou, 2019) give each skill an independent per-task value and sum it over the set. Datamodels (Ilyas et al., 2022) fits a regularized linear model that predicts execution success rates from binary indicators of which skills are selected. DeepSets (Zaheer et al., 2017) and Set Transformer (Lee et al., 2019) are black-box neural set regressors. We also include Linear Response, an ablation that sets $h(x) = x$ in Eq. (7). We fit every model using execution outcomes for all measured skill sets from the 11 training tasks and 20 randomly sampled skill sets from each of the 10 test tasks. We then evaluate each model's predictions on the remaining 2,806 task–set pairs from the test tasks. Appendix B.5 gives the implementation details and the training/test split.

Prediction. For a given task q, good models report higher scores for skill sets that are more likely to succeed on the frozen executor. We take every pair of skill sets from the same test task whose measured success rates differ by more than 0.25 and count how often the tested model ranks them correctly (Fig. 4, left). Our structured model in Eq. (7) achieves the highest ranking accuracy with the fewest parameters, exceeding DeepSets and Set Transformer by 0.026 and 0.044, respectively. The two neural baselines each use approximately 60 times as many parameters. Supplementary extrapolation and unseen-doc experiments test generalization to larger sets and new combinations of individually observed skills (Appendix B.5 and Fig. 6).

Selection. We further evaluate each model for skill selection. For each selection instance, every model selects the highest-scoring candidate skill set that satisfies the token budget. Figure 4 (right) reports the selected sets' mean success rate across 204 instances from eight test tasks, using executions held out from fitting. The dashed line marks the empirical ceiling, averaging the highest measured execution success rate among feasible sets in each instance. Appendix B.5 details the candidate pools,

rollout split, and averaging over three random seeds. Our structured value model achieves 98% of this ceiling and outperforms all baselines, supporting its effectiveness for skill selection on this testbed.

5.3 End-to-End Selection Effectiveness

The end-to-end test pairs the eight test tasks with budgets of 500, 750 and 1,000 tokens, giving 24 instances. It executes the skill set selected by each method on the frozen executor and measures success on executions that no fit or training used. The lookup tables of Section 5.2 cannot run on an unseen task, so we train a neural capability encoder that projects frozen `text-embedding-v4` (Zhang et al., 2025) embeddings of the task prompt and of each skill into 64 latent capability dimensions. We warm it up on annotated covering skill sets and then train it online on pass/fail feedback from the executor, on tasks other than the one being tested, and `BPS` selects on this encoder (Appendix B.7).

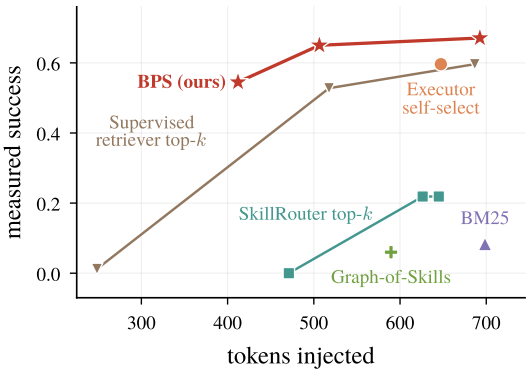

Figure 5: End-to-end skill selection: injected tokens versus measured execution success, averaged over the eight test tasks and three token budgets. From left to right, the stars show `BPS` with the neural capability encoder at $\rho = 2, 1, 0$, and each top-k curve shows $k = 1, 2, 3$.

We compare against the skill selectors in use today: BM25, the lexical retriever of the skill-retrieval literature (Su et al., 2026); the dense retriever our encoder embeds with, `text-embedding-v4`, fine-tuned on the same annotations and execution feedback as the encoder and ranking each document by its own score (Appendix B.7); SkillRouter (Zheng et al., 2026b), a trained retrieve-and-rerank router; Graph-of-Skills (Liu et al., 2026), which diffuses over an offline skill graph; and the executor selecting for itself from the skill catalogue. Both released systems run from their authors' code on our library. The two rankers are read at budgeted top-k, the first k documents in rank order that fit the budget, for $k = 1, 2, 3$. `BPS` has the matching control in a per-token price added to the fitted context sensitivity $\widehat{\kappa}_E$ of Eq. (7), a deployment setting that fixes where on the token–success trade-off it operates. We set this price to ρ/B, so that filling the budget costs an additional ρ at every budget, and report $\rho \in \{0, 1, 2\}$. Figure 5 plots the measured success of every rule against the tokens it injects.

Averaged over the three budgets, `BPS` attains the highest measured success at every token level (Fig. 5). Without the added price it reaches 0.67 on 693 tokens, and at $\rho = 1$ it still reaches 0.65 on only 506. The fine-tuned retriever, trained on the same supervision, reaches 0.53 at $k = 2$ on about as many tokens as `BPS` at $\rho = 1$ and needs 36% more to reach 0.60 at $k = 3$. The executor selecting for itself reaches 0.60 with 28% more, while the released routers and the lexical retriever stay at or below 0.22. Both rankers fail at $k = 1$ because every test task exercises at least two modules, and the top document covers only one. The routers and retrievers rarely select distractors but pick skills whose text matches the task, which need not cover the capabilities it exercises. `BPS` scores a set by the capabilities its documents jointly supply, and so covers what the task needs with fewer tokens.

6 Conclusion

We studied the skill selection problem for LLM agents under a hard token budget. We proposed a learnable model that captures complementary capability coverage, redundancy, and context cost. Based on this model, we developed `BPS`, a polynomial-time selection algorithm with a $(1 - 1/e,\ 1)$ bicriteria guarantee for the structured objective. On our controlled testbed, the structured model recovers skill-capability coverage and achieves the highest ranking accuracy among the baselines. With a learned capability encoder, `BPS` achieves higher execution success on held-out tasks with fewer injected tokens than a retriever trained on the same supervision and the executor's own selection. We discuss the limitations and future directions in Appendix C.

AI USE STATEMENT

In this work, we used generative AI tools for generating synthetic data sets (the private modules and skill documents of the testbed), implementing methods (the testbed, execution harness, fitting, and selection code), and providing feedback on the design of the experiments. We have not used generative AI tools for developing theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, proposing or refining hypotheses, or interpreting results, and assisting with translation, cleaning and reformatting data sets, and supporting qualitative and thematic data analysis are not applicable to this work. Additionally, we used generative AI tools for creating and modifying scientific figures, creating and editing software code, editing the paper to improve readability, and formatting references. We have reviewed all AI-assisted work. The authors read and tested all generated code, checked the selection code against the exact optimum on every instance (Appendix B.7), and checked every edited passage against the result it describes. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Appendix A gives the complete proof of Theorem 1; the pseudocode of `BPS` is Algorithm 1. Appendix B records the executor and execution protocol (Appendix B.1), the construction of the testbed and skill library (Appendix B.2), the fitting and evaluation of the value models (Appendices B.4 and B.5), the selection instances and rules (Appendix B.6), and the training of the neural encoder and the configuration of every deployed system (Appendix B.7). The testbed, skill library, execution records, and the code for fitting and selection will be released upon publication.

REFERENCES

Shengnan An et al. Skill-Based Few-Shot Selection for In-Context Learning. In *Proc. 2023 Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, pp. 13472–13492, 2023.

Anthropic. Equipping agents for the real world with agent skills. Anthropic Engineering Blog, 2025. URL https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills. Accessed 2026-07-23.

Hongcheol Cho, Ryangkyung Kang, and Youngeun Kim. SkillRet: A large-scale benchmark for skill retrieval in LLM agents. *arXiv preprint arXiv:2605.05726*, 2026.

Aniket Didolkar et al. Metacognitive Capabilities of LLMs: An Exploration in Mathematical Problem Solving. In *Adv. Neural Inf. Process. Syst. 37 (NeurIPS 2024)*, pp. 19783–19812, 2024.

Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

Moran Feldman. Guess free maximization of submodular and linear sums. *Algorithmica*, 83(3): 853–878, 2021.

Moran Feldman, Zeev Nutov, and Elad Shoham. Practical budgeted submodular maximization. *Algorithmica*, 85(5):1332–1371, 2023.

Michael Fore, Simranjit Singh, and Dimitrios Stamoulis. GeckOpt: LLM system efficiency via intent-based tool selection. In *Proc. Great Lakes Symp. VLSI (GLSVLSI)*, pp. 353–354, 2024.

Tiantian Gan and Qiyao Sun. RAG-MCP: Mitigating prompt bloat in LLM tool selection via retrieval-augmented generation. *arXiv preprint arXiv:2505.03275*, 2025.

Yudong Gao, Zongjie Li, Yuanyuan Yuan, Zimo Ji, Pingchuan Ma, and Shuai Wang. SkillReducer: Optimizing LLM agent skills for token efficiency. *arXiv preprint arXiv:2603.29919*, 2026.

Amirata Ghorbani and James Zou. Data Shapley: Equitable valuation of data for machine learning. In *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, volume 97, pp. 2242–2251, 2019.

Manu Ghulyani, Arunabh Singh, Karan Bharadwaj, Ankit Nath, and Suranjan Goswami. PACMS: Submodular context selection as a pluggable engine for LLM agents. *arXiv preprint arXiv:2606.20047*, 2026.

Negin Golrezaei, Karthikeyan Shanmugam, Alexandros G. Dimakis, Andreas F. Molisch, and Giuseppe Caire. FemtoCaching: Wireless video content delivery through distributed caching helpers. In *Proc. IEEE INFOCOM*, pp. 1107–1115, 2012.

Suning Gong, Qingqin Nong, Yue Wang, and Dingzhu Du. Budget-constrained profit maximization without non-negative objective assumption in social networks. *J. Glob. Optim.*, 90(4):1007–1030, 2024.

Qintian Guo, Chen Feng, Jieming Shi, Jing Tang, Xiaofang Zhou, and Sibo Wang. Efficient algorithms for budgeted profit maximization with theoretical guarantees. *IEEE Trans. Knowl. Data Eng.*, 38 (4):2234–2248, 2026.

Shivanshu Gupta, Matt Gardner, and Sameer Singh. Coverage-based example selection for in-context learning. In *Findings Assoc. Comput. Linguistics: EMNLP 2023*, pp. 13924–13950, 2023.

Christopher Harshaw, Moran Feldman, Justin Ward, and Amin Karbasi. Submodular maximization beyond non-negativity: Guarantees, fast algorithms, and applications. In *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, volume 97, pp. 2634–2643, 2019.

Andrew Ilyas, Sung Min Park, Logan Engstrom, Guillaume Leclerc, and Aleksander Madry. Datamodels: Understanding predictions with data and data with predictions. In *Proc. 39th Int. Conf. Mach. Learn. (ICML)*, volume 162, pp. 9525–9587, 2022.

Samir Khuller, Anna Moss, and Joseph Naor. The budgeted maximum coverage problem. *Inf. Process. Lett.*, 70(1):39–45, 1999.

Katrin Kirchhoff and Jeff Bilmes. Submodularity for data selection in machine translation. In *Proc. 2014 Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, pp. 131–141, 2014.

Ariel Kulik, Roy Schwartz, and Hadas Shachnai. A refined analysis of submodular greedy. *Oper. Res. Lett.*, 49(4):507–514, 2021.

Lilly Kumari, Shengjie Wang, Arnav Das, Tianyi Zhou, and Jeff Bilmes. An end-to-end submodular framework for data-efficient in-context learning. In *Findings Assoc. Comput. Linguistics: NAACL 2024*, pp. 3293–3308, 2024.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, volume 97, pp. 3744–3753, 2019.

Hao Li et al. Organizing, orchestrating, and benchmarking agent skills at ecosystem scale. *arXiv preprint arXiv:2603.02176*, 2026a.

Xiangyi Li et al. SkillsBench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*, 2026b.

Yanchao Li et al. SkillsInjector: Dynamic skill context construction for LLM agents. *arXiv preprint arXiv:2605.29794*, 2026c.

Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *Proc. 49th Annu. Meeting Assoc. Comput. Linguistics (ACL)*, pp. 510–520, 2011.

Dawei Liu et al. Graph-of-Skills: Dependency-aware structural retrieval for massive agent skills. In *Proc. 2026 Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2026.

Nelson F. Liu et al. Lost in the middle: How language models use long contexts. *Trans. Assoc. Comput. Linguistics*, 12:157–173, 2024.

Ali Modarressi, Hanieh Deilamsalehy, Franck Dernoncourt, Trung Bui, Ryan A. Rossi, Seunghyun Yoon, and Hinrich Schütze. NoLiMa: Long-context evaluation beyond literal matching. In *Proc. 42nd Int. Conf. Mach. Learn. (ICML)*, pp. 44554–44570, 2025.

Sofia Maria Nikolakaki, Alina Ene, and Evimaria Terzi. An efficient framework for balancing submodularity and cost. In *Proc. 27th ACM SIGKDD Conf. Knowl. Discov. Data Min.*, pp. 1256–1266, 2021. doi: 10.1145/3447548.3467367.

OpenAI. Build skills. Codex Documentation, 2026. URL https://learn.chatgpt.com/docs/build-skills. Accessed 2026-07-26.

Pierre Perrault, Jennifer Healey, Zheng Wen, and Michal Valko. On the approximation relationship between optimizing ratio of submodular (RS) and difference of submodular (DS) functions. *arXiv preprint arXiv:2101.01631*, 2021.

Konstantinos Poularakis, Jaime Llorca, Antonia M. Tulino, Ian Taylor, and Leandros Tassiulas. Joint service placement and request routing in multi-cell mobile edge computing networks. In *Proc. IEEE INFOCOM*, pp. 10–18, 2019.

Yujia Qin et al. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *Proc. 12th Int. Conf. Learn. Represent. (ICLR)*, 2024.

Changle Qu et al. Towards completeness-oriented tool retrieval for large language models. In *Proc. ACM Int. Conf. Inf. Knowl. Manage. (CIKM)*, pp. 1930–1940, 2024.

Freda Shi et al. Large language models can be easily distracted by irrelevant context. In *Proc. 40th Int. Conf. Mach. Learn. (ICML)*, volume 202, pp. 31210–31227, 2023.

Weihang Su et al. Skill retrieval augmentation for agentic AI. *arXiv preprint arXiv:2604.24594*, 2026.

Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Oper. Res. Lett.*, 32(1):41–43, 2004.

Karan Vombatkere and Evimaria Terzi. Approximate Pareto frontiers for submodular utility and cost tradeoffs. In *Proc. 35th ACM Int. Conf. Inf. Knowl. Manag. (CIKM)*, 2026. doi: 10.1145/3799682.3841064.

Zifei Wang, Wei Wen, Qiang Ji, Keyu Chen, Ruizhi Qiao, and Xing Sun. Skill is not document: Query-conditioned compatibility for LLM agent skill routing. *arXiv preprint arXiv:2606.03565*, 2026.

Jinying Xiao et al. SkillSight: Calibrating generic content bias for skill retrieval. *arXiv preprint arXiv:2607.18785*, 2026.

Zifan Xu, Haozhu Wang, Dmitriy Bespalov, Xian Wu, Peter Stone, and Yanjun Qi. LaRS: Latent Reasoning Skills for Chain-of-Thought Reasoning. In *Findings Assoc. Comput. Linguistics: EMNLP 2024*, pp. 3624–3643, 2024.

An Yang et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Cehao Yang et al. A survey of agent skills: Toward procedural infrastructure for LLM agents. *Preprints*, 2026. doi: 10.20944/preprints202605.1276.v1.

Grigory Yaroslavtsev, Samson Zhou, and Dmitrii Avdiukhin. "Bring your own greedy"+Max: Near-optimal 1/2-approximations for submodular knapsack. In *Proc. 23rd Int. Conf. Artif. Intell. Statist. (AISTATS)*, pp. 3263–3274, 2020.

Jiacheng Ye, Zhiyong Wu, Jiangtao Feng, Tao Yu, and Lingpeng Kong. Compositional exemplars for in-context learning. In *Proc. 40th Int. Conf. Mach. Learn. (ICML)*, volume 202, pp. 39818–39833, 2023.

Michelle Yuan, Khushbu Pahwa, Shuaichen Chang, Mustafa Devrim Kaba, Jiarong Jiang, Xiaofei Ma, Yi Zhang, and MONICA SUNKARA. Automated composition of agents: A knapsack approach for agentic component selection. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=1LPPMAUlaT.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep sets. In *Adv. Neural Inf. Process. Syst. (NeurIPS)*, volume 30, pp. 3391–3401, 2017.

Kun Zeng et al. Group of skills: Group-structured skill retrieval for agent skill libraries. *arXiv preprint arXiv:2605.06978*, 2026.

Hongyang Zhang and Wenchang Luo. A streaming algorithm for non-monotone regularized submodular maximization. *Oper. Res. Lett.*, 67:107456, 2026.

Yanzhao Zhang et al. Qwen3 Embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025.

Xinyu Zhao et al. Generative skill composition for LLM agents. *arXiv preprint arXiv:2606.32025*, 2026.

Jingyuan Zheng et al. SkillSelect-Serve: QoS-aware budgeted skill service recommendation for LLM agents. *arXiv preprint arXiv:2607.00011*, 2026a.

YanZhao Zheng et al. SkillRouter: Skill routing for LLM agents at scale. *arXiv preprint arXiv:2603.22455*, 2026b.

Terry Yue Zhuo et al. BigCodeBench: Benchmarking code generation with diverse function calls and complex instructions. In *Proc. 13th Int. Conf. Learn. Represent. (ICLR)*, 2025.

A Proof of Theorem 1

Throughout this section, fix the query q, executor E, and budget B. For brevity, we write $\widehat{G}(S) = \widehat{G}_E(q, S)$, $\widehat{F}(S) = \widehat{F}_E(q, S)$, and $\widehat{\kappa} = \widehat{\kappa}_E$, and let $\alpha = 1 - 1/e$.

Proof sketch. We first outline the argument, then give it in full. Fix any feasible T. If $|T| \le 3$, the seed enumeration of Algorithm 1 alone yields the claim. Otherwise the argument has three steps.

(i) Order T greedily by marginal benefit and take the first two items in this greedy order as the seed J, which Algorithm 1 enumerates; the residual function $f(U) := \widehat{G}(J \cup U) - \widehat{G}(J)$ is again monotone submodular.

(ii) Following (Kulik et al., 2021), trace the residual benefit of the density chain grown from J as a piecewise-affine trajectory $V(u)$ of the added length u, and lower-bound it by a piecewise-exponential function φ. Evaluating at r, the length of $T \setminus J$ minus its longest item, gives $\widehat{G}(J) + V(r) \ge \alpha\, \widehat{G}(T)$; removing the longest item is what keeps every remaining item of T feasible while the chain runs up to r.

(iii) *Budget-aligned interpolation.* The point $u = r$ generally falls strictly inside a chain segment, so $V(r)$ is a convex combination of the residual benefits of two consecutive recorded prefixes. Because the penalty $\widehat{\kappa}\,\ell(\cdot)$ is linear in the same length coordinate, the same combination applied to $\widehat{F}$ equals $\widehat{G}(J) + V(r) - \widehat{\kappa}(\ell(J) + r) \ge \alpha\, \widehat{G}(T) - \widehat{\kappa}\,\ell(T)$, so one of the two prefixes attains the bound. Since BPS returns the $\widehat{F}$-maximizer over all recorded prefixes, the bound holds for its single output and every T. Step (iii) handles the non-monotonicity of $\widehat{F}$: the chain endpoint need not attain the required bound. Because the penalty is linear in length, the bound at $u = r$ transfers to one of the two adjacent recorded prefixes.

A.1 Seed construction and residual function

We first write $\widehat{G}(i \mid S) = \widehat{G}(S \cup \{i\}) - \widehat{G}(S)$ for the marginal benefit. We order the items of T greedily by nonincreasing marginal benefit with an arbitrary fixed tie-breaking rule:

$$\begin{aligned} t_j &\in \operatorname*{argmax}_{i \in T \setminus \{t_1, \cdots, t_{j-1}\}} \widehat{G}\left(i \mid \{t_1, \cdots, t_{j-1}\}\right), \\ m_j &:= \widehat{G}\left(t_j \mid \{t_1, \cdots, t_{j-1}\}\right), \quad \forall j \in [|T|]. \end{aligned} \tag{10}$$

If $|T| \le 2$, then Algorithm 1 enumerates T itself as a seed, and we have $\widehat{F}(T) = \widehat{G}(T) - \widehat{\kappa}\ell(T) \ge \alpha\widehat{G}(T) - \widehat{\kappa}\ell(T)$.

We set the feasible seed set J with $\ell(J) \le B$ as

$$J := \{t_1, t_2\}. \tag{11}$$

Then Algorithm 1 enumerates it in Line 2 and records it in Line 4, so $J \in \mathcal{P}$. If $|T| = 3$: $\widehat{G}(J) = m_1 + m_2 \ge \frac{2}{3}(m_1 + m_2 + m_3) = \frac{2}{3}\widehat{G}(T) \ge \alpha\widehat{G}(T)$ and $\ell(J) \le \ell(T)$ give $\widehat{F}(J) \ge \alpha\, \widehat{G}(T) - \widehat{\kappa}\,\ell(T)$.

In the following, we assume $|T| \ge 4$. Our goal is to prove that on the chain grown from J, some recorded prefix S_T satisfies $\widehat{F}(S_T) \ge \alpha\widehat{G}(T) - \widehat{\kappa}\ell(T)$.

Inspired by the analysis in (Sviridenko, 2004), we define the residual function

$$f(U) := \widehat{G}(J \cup U) - \widehat{G}(J), \quad \forall U \subseteq [L]. \tag{12}$$

By the properties of $\widehat{G}$, f is normalized ($f(\varnothing) = 0$), nonnegative, nondecreasing, and submodular. Let $P = T \setminus J$ denote the comparator residual. For every $v \in P$, we have $f(\{v\}) = \widehat{G}(\{v\} \cup J) - \widehat{G}(J) = \widehat{G}(v \mid J) \le \widehat{G}(v \mid \{t_1\}) \le m_2$, where the first inequality uses submodularity and the second uses the greedy ordering Eq. (10). Therefore, we have

$$\widehat{G}(J) = \widehat{G}(\{t_1, t_2\}) - \widehat{G}(\{t_1\}) + \widehat{G}(\{t_1\}) = m_1 + m_2 \ge 2m_2 \ge 2f(\{v\}), \quad \forall v \in P. \tag{13}$$

We choose $v^\star \in P$ as the item with maximum token length, i.e., $v^\star \in \operatorname{argmax}_{v \in P} \ell(\{v\})$. Set

$$r = \ell(P \setminus \{v^\star\}). \tag{14}$$

Then we can show that the density chain starting with seed J must have total token length larger than r.

A.2 BOUNDING-FUNCTION DOMINATION

Run the density chain of Algorithm 1 from seed J, and let $A_0 = \varnothing \subsetneq A_1 \subsetneq \cdots \subsetneq A_m$ be its accepted additions, so the actual recorded prefixes are $J \cup A_j$. Write $\{a_j\} = A_j \setminus A_{j-1}$ for the j-th accepted item. Following (Kulik et al., 2021), we define the piecewise-affine residual-benefit trajectory V by $V(0) = 0$ and, for $\ell(A_{j-1}) \le u \le \ell(A_j)$,

$$V(u) := f(A_{j-1}) + \big(u - \ell(A_{j-1})\big)\frac{f(a_j \mid A_{j-1})}{\ell(\{a_j\})}. \tag{15}$$

The function $V : [0, \ell(A_m)] \to \mathbb{R}_{\ge 0}$ traces the residual benefit accumulated by the density chain as a function of the total added length u beyond the seed J. At each breakpoint $u = \ell(A_j)$, the trajectory evaluates to $V(\ell(A_j)) = f(A_j)$, the exact residual benefit of the j-th recorded prefix. Between consecutive breakpoints, V interpolates linearly. Then we have the following lemmas.

Lemma 1 (Trajectory coverage). *$V(r)$ is well-defined; that is, $\ell(A_m) \ge r$.*

Proof. If $P \subseteq A_m$, then $\ell(A_m) \ge \ell(P) > r$ by definition in Eq. (14). Otherwise pick $v \in P \setminus A_m$. When the chain stops, adding v is infeasible. Then $\ell(A_m) + \ell(\{v\}) > B - \ell(J)$. Since $\ell(\{v\}) \le \ell(\{v^\star\})$ and $\ell(T) \le B$, we have $\ell(A_m) > B - \ell(J) - \ell(\{v\}) \ge \ell(T) - \ell(J) - \ell(\{v^\star\}) = r$. □

Lemma 2 (Residual feasibility before r). *Consider an accepted segment whose left endpoint $u_0 = \ell(A_{j-1})$ satisfies $u_0 < r$. Then every item $v \in P \setminus A_{j-1}$ is feasible at that point.*

Proof. By maximality of $\ell(\{v^\star\})$, we have $\ell(A_{j-1} \cup \{v\}) = u_0 + \ell(\{v\}) < r + \ell(\{v^\star\}) = \ell(P) = \ell(T) - \ell(J) \le B - \ell(J)$. Therefore, v is feasible when the algorithm selects a_j. □

Lemma 2 is where removing the longest residual item pays off.

We adapt the bounding-function technique of (Kulik et al., 2021). Partition P into two nonempty blocks $\{v^\star\}$ and $R = P \setminus \{v^\star\}$, and order them as (X_1, X_2) so that

$$\frac{f(X_1)}{\ell(X_1)} \ge \frac{f(X_2)}{\ell(X_2)}. \tag{16}$$

Set $d_j = \ell(X_j)$ for $j = 1, 2$, and define the block densities

$$\rho_1 = \frac{f(X_1)}{d_1}, \quad \rho_2 = \frac{f(X_2 \mid X_1)}{d_2}, \tag{17}$$

where $f(X_2 \mid X_1) = f(X_1 \cup X_2) - f(X_1)$. Submodularity and Eq. (16) give $\rho_1 \ge \rho_2$. Write

$$D = \ell(P) = d_1 + d_2. \tag{18}$$

If $\rho_2 > 0$, let $D_1 = d_1 \ln \frac{\rho_1}{\rho_2}$. If $\rho_2 = 0$, set $D_1 = +\infty$. Define a continuous function $\varphi : [0, \infty) \to \mathbb{R}_{\ge 0}$ by

$$\varphi(u) = \begin{cases} f(X_1)\,(1 - \exp(-u/d_1)), & 0 \le u < D_1, \\ f(P) - \rho_2 D \exp\left(-\dfrac{u - D_1}{D}\right), & u \ge D_1. \end{cases} \tag{19}$$

The first branch of φ approaches $f(X_1)$ at exponential rate $1/d_1$, while the second approaches $f(P)$ at rate $1/D$. When D_1 is finite, the two expressions agree there.

To use φ in the proof, we need to relate it both to the greedy trajectory and to the comparator's total benefit. Lemma 3 shows that $V(u) \ge \varphi(u)$ throughout $[0, r]$, making φ a lower bound on the residual benefit along the trajectory. Lemma 4 then shows that the seed benefit together with $\varphi(r)$ is at least $\alpha\widehat{G}(T)$. Combining the two gives $\widehat{G}(J) + V(r) \ge \alpha\widehat{G}(T)$, the benefit bound that the next subsection converts into a bound on the fitted objective at a recorded prefix. We prove these lemmas in Appendices A.5 and A.6.

Lemma 3 (Trajectory bound). $V(u) \ge \varphi(u), \forall u \in [0, r]$.

Lemma 4 (Static bound at r). $\widehat{G}(J) + \varphi(r) \ge \alpha\widehat{G}(T)$.

A.3 BUDGET-ALIGNED INTERPOLATION

Lemmas 3 and 4 give

$$\widehat{G}(J) + V(r) \geq \widehat{G}(J) + \varphi(r) \geq \alpha\, \widehat{G}(T). \tag{20}$$

Choose consecutive recorded prefixes A_{j-1}, A_j whose segment contains r. There exists $\lambda \in [0,1]$ with $V(r) = (1-\lambda)\, f(A_{j-1}) + \lambda\, f(A_j)$, and $r = (1-\lambda)\, \ell(A_{j-1}) + \lambda\, \ell(A_j)$. Because the penalty $\widehat{\kappa}\, \ell(\cdot)$ is the same linear function of the same length coordinate, the two interpolations combine: $(1-\lambda)\widehat{F}(J \cup A_{j-1}) + \lambda \widehat{F}(J \cup A_j) = \widehat{G}(J) + V(r) - \widehat{\kappa}(\ell(J) + r)$. Since $\ell(J) + r = \ell(T) - \ell(\{v^\star\}) \leq \ell(T)$, Eq. (20) gives $(1-\lambda)\widehat{F}(J \cup A_{j-1}) + \lambda \widehat{F}(J \cup A_j) \geq \alpha \widehat{G}(T) - \widehat{\kappa}(\ell(T) - \ell(\{v^\star\})) \geq \alpha \widehat{G}(T) - \widehat{\kappa}\ell(T)$. A convex combination of two numbers is at most their maximum, so at least one of the two recorded prefixes satisfies

$$\widehat{F}(J \cup A_s) \geq \alpha \widehat{G}(T) - \widehat{\kappa}\ell(T), \quad s \in \{j-1, j\}. \tag{21}$$

Therefore, in every case, some recorded candidate $S_T \in \mathcal{P}$ satisfies $\widehat{F}(S_T) \geq \alpha\, \widehat{G}(T) - \widehat{\kappa}\, \ell(T)$. Since $S_{\texttt{BPS}}$ maximizes $\widehat{F}$ over $\mathcal{P}$ (Line 11 of Algorithm 1), $\widehat{F}(S_{\texttt{BPS}}) \geq \widehat{F}(S_T)$. The seed and witness prefix depend on T, but every seed of size at most two is enumerated and one selection maximizes $\widehat{F}$ over all recorded candidates, so the single output $S_{\texttt{BPS}}$ satisfies Eq. (9) for every $T \in \mathcal{F}_B$. This statement finishes the proof of Theorem 1. ■

A.4 AUXILIARY LEMMAS

Throughout the remaining subsections we use the notation introduced above: the seed $J = \{t_1, t_2\}$ of Eq. (11), the residual function f of Eq. (12), the comparator residual $P = T \setminus J$ with $|T| \geq 4$, the longest residual item $v^\star$, the remainder $R = P \setminus \{v^\star\}$, the lengths $p := \ell(\{v^\star\})$ and $r = \ell(R)$ of Eq. (14), the block order (X_1, X_2) of Eq. (16) with $d_j = \ell(X_j)$ and densities $\rho_1 = f(X_1)/d_1$, $\rho_2 = f(X_2 \mid X_1)/d_2$, the total residual length $D = \ell(P) = d_1 + d_2$ of Eq. (18), the crossover point D_1, the bounding function φ of Eq. (19), the trajectory V of Eq. (15), and $\alpha = 1 - 1/e$. Since $|T| \geq 4$ we have $|P| \geq 2$, so both blocks $\{v^\star\}$ and R are nonempty; all item lengths are positive, hence

$$p > 0, \qquad r > 0, \qquad d_1 > 0, \qquad d_2 > 0, \qquad D = p + r. \tag{22}$$

Recall from Appendix A.1 that f is normalized ($f(\varnothing) = 0$), nonnegative, nondecreasing, and submodular, and that submodularity with Eq. (16) gives $\rho_1 \geq \rho_2 \geq 0$.

We first record two auxiliary lemmas whose proofs are elementary computations; both are used repeatedly below.

Lemma 5 (Residual marginals and the trajectory)**.**

(i) For every $U \subseteq [L]$ and $v \in [L]$, $f(v \mid U) = \widehat{G}(v \mid J \cup U) \geq 0$.

(ii) V is continuous, piecewise affine, and nondecreasing on $[0, \ell(A_m)]$; it satisfies $V(\ell(A_j)) = f(A_j)$ for every j, and it is defined on all of $[0, r]$.

(iii) Consider the chain step that selects a_j from the state $J \cup A_{j-1}$, and let $\sigma_j = f(a_j \mid A_{j-1})/\ell_{a_j}$ denote its density. Then $f(v \mid A_{j-1})/\ell(\{v\}) \leq \sigma_j$ for every item $v \notin J \cup A_{j-1}$ that is feasible at that step.

Proof. (i) Expanding definition Eq. (12) twice,

$$f(v \mid U) = f(U \cup \{v\}) - f(U) = \big(\widehat{G}(J \cup U \cup \{v\}) - \widehat{G}(J)\big) - \big(\widehat{G}(J \cup U) - \widehat{G}(J)\big) = \widehat{G}(v \mid J \cup U),$$

which is nonnegative because $\widehat{G}$ is nondecreasing.

(ii) By construction Eq. (15), V is affine on each segment $[\ell(A_{j-1}), \ell(A_j)]$ with slope $\sigma_j = f(a_j \mid A_{j-1})/\ell_{a_j}$, and its value at the right endpoint of segment j telescopes to $V(\ell(A_j)) = f(A_{j-1}) + f(a_j \mid A_{j-1}) = f(A_j)$, which equals the value used at the left endpoint of segment $j+1$; hence V is continuous. Each slope satisfies $\sigma_j \geq 0$ by (i) and $\ell_{a_j} > 0$, so V is nondecreasing. Finally, $\ell(A_m) \geq r$ by Lemma 1, so $[0, r]$ lies in the domain of V.

(iii) Line 6 of Algorithm 1 selects, among all items $i \notin J \cup A_{j-1}$ that are feasible at the current state $S = J \cup A_{j-1}$, one maximizing $\widehat{G}(i \mid S)/\ell_i$. By (i), $\widehat{G}(v \mid J \cup A_{j-1}) = f(v \mid A_{j-1})$ for every such v, so maximizing the density with respect to $\widehat{G}$ is the same as maximizing it with respect to f; in particular the selected item a_j satisfies $f(v \mid A_{j-1})/\ell(\{v\}) \le f(a_j \mid A_{j-1})/\ell_{a_j} = \sigma_j$ for every feasible v. □

Lemma 6 (Properties of the bounding function)**.** *The function φ of Eq. (19) satisfies:*

(i) $\varphi(0) = 0$*; this holds on the first branch when* $D_1 > 0$ *and on the second branch when* $D_1 = 0$*.*

(ii) φ *is continuous on* $[0, \infty)$*. If* $0 < D_1 < +\infty$*, the two branches agree at* $u = D_1$*.*

(iii) On the first branch $(0, D_1)$*,* $\varphi'(u) = \big(f(X_1) - \varphi(u)\big)/d_1$*; on the second branch* (D_1, ∞)*,* $\varphi'(u) = \big(f(P) - \varphi(u)\big)/D$*.*

Proof. We use throughout the chain-rule decomposition

$$f(P) = f(X_1) + f(X_2 \mid X_1) = f(X_1) + \rho_2 d_2 = \rho_1 d_1 + \rho_2 d_2. \tag{23}$$

(i) If $D_1 > 0$, the first branch gives $\varphi(0) = f(X_1)(1 - e^0) = 0$. If $D_1 = 0$, then by definition of D_1 we have $\rho_2 > 0$ and $d_1 \ln(\rho_1/\rho_2) = 0$, hence $\rho_1 = \rho_2$; the second branch then gives, using Eq. (23), $\varphi(0) = f(P) - \rho_2 D e^0 = \rho_1 d_1 + \rho_2 d_2 - \rho_2(d_1 + d_2) = 0$.

(ii) Let $0 < D_1 < +\infty$, so $\rho_2 > 0$ and $e^{-D_1/d_1} = \rho_2/\rho_1$. The left limit at D_1 along the first branch is

$$f(X_1)\big(1 - e^{-D_1/d_1}\big) = f(X_1)\Big(1 - \frac{\rho_2}{\rho_1}\Big) = f(X_1) - \rho_2 d_1,$$

where the last step uses $f(X_1) = \rho_1 d_1$. The value at D_1 on the second branch is $f(P) - \rho_2 D e^0 = f(P) - \rho_2(d_1 + d_2)$, which equals $f(X_1) - \rho_2 d_1$ by Eq. (23). The two branches agree, and each branch is continuous, so φ is continuous. If $D_1 = 0$ or $D_1 = +\infty$, then φ is defined on all of $[0, \infty)$ by the second or first exponential branch, respectively, and is therefore continuous in these cases as well.

(iii) On $(0, D_1)$, differentiating $\varphi(u) = f(X_1)(1 - e^{-u/d_1})$ gives $\varphi'(u) = \frac{f(X_1)}{d_1} e^{-u/d_1}$, while $f(X_1) - \varphi(u) = f(X_1) e^{-u/d_1}$; dividing by d_1 matches. On (D_1, ∞), differentiating $\varphi(u) = f(P) - \rho_2 D \exp\big(-\frac{u - D_1}{D}\big)$ gives $\varphi'(u) = \rho_2 \exp\big(-\frac{u - D_1}{D}\big)$, while $f(P) - \varphi(u) = \rho_2 D \exp\big(-\frac{u - D_1}{D}\big)$; dividing by D matches. □

A.5 Proof of Lemma 3

Proof. We first bound the density on each segment, then compare V and φ using differential inequalities and an integrating factor.

Step 1 (density lower bound). Fix an accepted segment starting at A_{j-1} with left endpoint $u_0 = \ell(A_{j-1}) < r$, and let $\sigma_j = f(a_j \mid A_{j-1})/\ell_{a_j}$ be the density of the item selected by Algorithm 1 on that segment. Let $Q \subseteq P$ be any nonempty subset such that every member of $Q \setminus A_{j-1}$ is feasible at this step; by Lemma 2, this holds for *every* $Q \subseteq P$, since $u_0 < r$. We claim

$$f(Q) \;\le\; f(A_{j-1} \cup Q) \;\le\; f(A_{j-1}) + \sum_{v \in Q \setminus A_{j-1}} f(v \mid A_{j-1}) \;\le\; f(A_{j-1}) + \sigma_j\, \ell(Q). \tag{24}$$

The first inequality is monotonicity of f. For the second, enumerate $Q \setminus A_{j-1} = \{v_1, \dots, v_s\}$ in an arbitrary order and telescope:

$$f(A_{j-1} \cup Q) - f(A_{j-1}) = \sum_{k=1}^{s} f\big(v_k \mid A_{j-1} \cup \{v_1, \dots, v_{k-1}\}\big) \le \sum_{k=1}^{s} f(v_k \mid A_{j-1}),$$

where each summand is bounded via submodularity of f (conditioning on a superset of A_{j-1} can only decrease the marginal). For the third inequality, every $v \in Q \setminus A_{j-1}$ is feasible at this step, so

Lemma 5(iii) gives $f(v \mid A_{j-1}) \le \sigma_j \ell(\{v\})$; summing over $Q \setminus A_{j-1}$ and using $\sigma_j \ge 0$ together with $\ell(Q \setminus A_{j-1}) \le \ell(Q)$ (lengths are positive) yields the claim. Rearranging Eq. (24) and using $\ell(Q) > 0$,

$$\sigma_j \;\ge\; \frac{f(Q) - f(A_{j-1})}{\ell(Q)} \;\ge\; \frac{f(Q) - V(u)}{\ell(Q)} \qquad \text{for every } u \in [\ell(A_{j-1}), \ell(A_j)], \tag{25}$$

where the second inequality holds because V is nondecreasing (Lemma 5(ii)) with $V(\ell(A_{j-1})) = f(A_{j-1})$, so $V(u) \ge f(A_{j-1})$ on the whole segment. We apply Eq. (25) with $Q = X_1$ and $Q = P$ on each segment starting before r.

Step 2 (differential inequality on each branch). Consider the difference $W(u) := V(u) - \varphi(u)$ on $[0, r]$. Both V and φ are continuous (Lemma 5(ii) and Lemma 6(ii)), so W is continuous. Partition $[0, \min(r, D_1)]$ (and, when $D_1 < r$, also $[D_1, r]$) by the finitely many trajectory breakpoints $\ell(A_0) < \ell(A_1) < \cdots$ falling in the respective interval. On the interior of each cell of this partition, V is affine with $V'(u) = \sigma_j$ for the segment index j containing the cell, and φ is differentiable (Lemma 6(iii)).

First branch. Let u lie in the interior of a cell of $[0, \min(r, D_1)]$. The segment containing u has left endpoint $u_0 \le u < r$, so Eq. (25) applies with $Q = X_1$, and Lemma 6(iii) gives the ODE for φ. Subtracting,

$$W'(u) = V'(u) - \varphi'(u) \;\ge\; \frac{f(X_1) - V(u)}{d_1} - \frac{f(X_1) - \varphi(u)}{d_1} \;=\; -\frac{W(u)}{d_1}. \tag{26}$$

Second branch. Let $D_1 < r$ and let u lie in the interior of a cell of $[D_1, r]$. Again the segment containing u has left endpoint $u_0 < r$, so Eq. (25) applies with $Q = P$, and the same subtraction gives

$$W'(u) \;\ge\; \frac{f(P) - V(u)}{D} - \frac{f(P) - \varphi(u)}{D} \;=\; -\frac{W(u)}{D}. \tag{27}$$

Step 3 (integrating factor and propagation across breakpoints). Fix a branch and write ℓ_Q for its length scale ($\ell_Q = d_1$ on the first branch, $\ell_Q = D$ on the second). On the interior of each cell,

$$\frac{\mathrm{d}}{\mathrm{d}u}\Big(e^{u/\ell_Q}\, W(u)\Big) = e^{u/\ell_Q}\Big(W'(u) + \frac{W(u)}{\ell_Q}\Big) \;\ge\; 0 \tag{28}$$

by Eq. (26) or Eq. (27). Hence $e^{u/\ell_Q} W(u)$ is nondecreasing on the interior of each cell; since it is continuous on the whole branch interval and the partition has finitely many cells, it is nondecreasing on the entire branch interval. Therefore, if $W \ge 0$ at the left endpoint of the branch interval, then $e^{u/\ell_Q} W(u) \ge 0$, hence $W(u) \ge 0$, throughout that interval.

By Eq. (15) and Lemma 6(i), $W(0) = 0$. If $D_1 \ge r$, including $D_1 = +\infty$ when $\rho_2 = 0$, the first-branch argument gives $W \ge 0$ on $[0, r]$; continuity includes the endpoint when $D_1 = r$. If $D_1 = 0$, the second-branch argument applies from $u = 0$. Finally, if $0 < D_1 < r$, the first-branch argument and continuity give $W(D_1) \ge 0$, which provides the initial value for the second branch. Thus $V(u) \ge \varphi(u)$ for all $u \in [0, r]$, proving Lemma 3. □

A.6 Proof of Lemma 4

We use the following standard inequality; we include its one-line proof for completeness.

Lemma 7 (Log-sum inequality)**.** *For positive reals a_1, a_2, b_1, b_2,*

$$a_1 \ln \frac{b_1}{a_1} + a_2 \ln \frac{b_2}{a_2} \;\le\; (a_1 + a_2) \ln \frac{b_1 + b_2}{a_1 + a_2}. \tag{29}$$

Proof. Apply Jensen's inequality to the concave function $\ln$ with weights $\lambda_i = a_i/(a_1 + a_2)$ and points $u_i = b_i/a_i$:

$$\frac{a_1}{a_1 + a_2} \ln \frac{b_1}{a_1} + \frac{a_2}{a_1 + a_2} \ln \frac{b_2}{a_2} \;\le\; \ln\Big(\frac{a_1}{a_1 + a_2} \cdot \frac{b_1}{a_1} + \frac{a_2}{a_1 + a_2} \cdot \frac{b_2}{a_2}\Big) = \ln \frac{b_1 + b_2}{a_1 + a_2}.$$

Multiplying both sides by $a_1 + a_2 > 0$ gives Eq. (29). □

Proof of Lemma 4. Abbreviate $C := \widehat{G}(J)$ and $H := \widehat{G}(T)$ for the duration of this proof. Since $J \cup P = T$, definition Eq. (12) gives

$$f(P) = \widehat{G}(T) - \widehat{G}(J) = H - C. \tag{30}$$

Step 0 (reduction and common identities). If $f(P) = 0$, then $f(X_1) = 0$ by nonnegativity and monotonicity of f. Thus $\rho_2 = 0$ and $D_1 = +\infty$, so Eq. (19) gives $\varphi \equiv 0$. Together with Eq. (30), this yields

$$C + \varphi(r) = C = H \geq \alpha H.$$

Assume from now on that $f(P) > 0$. Then $\rho_1 > 0$: otherwise $f(X_1) = 0$, and the block order Eq. (16) forces $f(X_2) \leq \frac{d_2}{d_1} f(X_1) = 0$, whence by submodularity $f(X_2 \mid X_1) \leq f(X_2) = 0$ and $f(P) = f(X_1) + f(X_2 \mid X_1) \leq 0$, a contradiction.

Define (x, z) according to the block order:

$$(x, z) = \begin{cases} \big(f(\{v^\star\}),\ f(R \mid \{v^\star\})\big), & X_1 = \{v^\star\}, \\ \big(f(v^\star \mid R),\ f(R)\big), & X_1 = R. \end{cases} \tag{31}$$

In both orders the chain rule gives $x + z = f(P)$, so by Eq. (30),

$$H = C + x + z. \tag{32}$$

In both orders we also have

$$C \geq 2x. \tag{33}$$

Indeed, if $X_1 = \{v^\star\}$ then $x = f(\{v^\star\})$ and Eq. (13) applies directly with $v = v^\star$; if $X_1 = R$ then submodularity gives $x = f(v^\star \mid R) \leq f(\{v^\star\})$, and Eq. (13) gives $C \geq 2f(\{v^\star\}) \geq 2x$.

Finally, recall the block data in the two orders. If $X_1 = \{v^\star\}$: $d_1 = p$, $d_2 = r$, $\rho_1 = x/p$, $\rho_2 = z/r$. If $X_1 = R$: $d_1 = r$, $d_2 = p$, $\rho_1 = z/r$, $\rho_2 = x/p$. In both orders, whenever $x, z > 0$,

$$d_1 \ln \rho_1 + d_2 \ln \rho_2 \ = \ p \ln \frac{x}{p} + r \ln \frac{z}{r}, \tag{34}$$

because the two summands on the left are exactly the two summands on the right, possibly in the opposite order.

We now distinguish the same three structural cases as the refined analysis of (Kulik et al., 2021); Step 4 verifies that they are exhaustive.

Step 1 (Case 1: $r \geq D_1$). Since $r < \infty$, the case condition forces $D_1 < \infty$, hence $\rho_2 > 0$ by the definition of D_1, and with $\rho_1 \geq \rho_2$ both block values are positive; reading off the block data above, this means

$$x > 0 \quad \text{and} \quad z > 0$$

in both orders, so every logarithm below has a strictly positive argument. Because $r \geq D_1$, the second branch of Eq. (19) evaluates $\varphi(r)$, and by Eq. (30) and Eq. (32),

$$C + \varphi(r) = C + f(P) - \rho_2 D \exp\Big(-\frac{r - D_1}{D}\Big) = H - \rho_2 D \exp\Big(-\frac{r - D_1}{D}\Big). \tag{35}$$

It therefore suffices to bound the exponential tail by H/e.

First, combining Eq. (33) with Eq. (32),

$$H - z = C + x \geq 2x + x = 3x. \tag{36}$$

Next we bound the weighted-log expression Eq. (34). Using $x \leq (H - z)/3$ from Eq. (36) and the monotonicity of $\ln$, then splitting off $\ln 3$, then applying Lemma 7 with $(a_1, a_2) = (p, r)$ and $(b_1, b_2) = (H - z, z)$ (all four are positive, $p + r = D$, and $(H - z) + z = H$), and finally using $\ln 3 > 1$ with $p > 0$:

$$\begin{aligned} p \ln \frac{x}{p} + r \ln \frac{z}{r} &\leq p \ln \frac{H - z}{3p} + r \ln \frac{z}{r} \\ &= -p \ln 3 + p \ln \frac{H - z}{p} + r \ln \frac{z}{r} \\ &\leq -p \ln 3 + D \ln \frac{H}{D} \\ &\leq -p + D \ln \frac{H}{D}. \end{aligned} \tag{37}$$

Now rewrite the exponential tail. Using $r = D - p$,

$$\rho_2 D \exp\Big(-\frac{r - D_1}{D}\Big) = D \exp\Big(\ln \rho_2 - \frac{D - p - D_1}{D}\Big) = D \exp\Big(-1 + \frac{p + D_1}{D} + \ln \rho_2\Big),$$

and since $D_1 = d_1 \ln(\rho_1/\rho_2)$ and $D = d_1 + d_2$,

$$\frac{D_1}{D} + \ln \rho_2 = \frac{d_1 \ln \rho_1 - d_1 \ln \rho_2 + D \ln \rho_2}{D} = \frac{d_1 \ln \rho_1 + d_2 \ln \rho_2}{D},$$

so that, by Eq. (34),

$$\rho_2 D \exp\Big(-\frac{r - D_1}{D}\Big) = D \exp\Big(-1 + \frac{p + p \ln \frac{x}{p} + r \ln \frac{z}{r}}{D}\Big). \tag{38}$$

Substituting the bound Eq. (37) into Eq. (38) and using the monotonicity of $\exp$,

$$\rho_2 D \exp\Big(-\frac{r - D_1}{D}\Big) \;\le\; D \exp\Big(-1 + \frac{D \ln(H/D)}{D}\Big) = D \cdot \frac{1}{e} \cdot \frac{H}{D} = \frac{H}{e}. \tag{39}$$

Combining Eq. (35) and Eq. (39),

$$C + \varphi(r) \;\ge\; H - \frac{H}{e} \;=\; \alpha H.$$

The boundary subcases are included: $r = D_1$ uses the second branch of Eq. (19), which is defined for all $u \ge D_1$, and $D_1 = 0$ poses no difficulty since only the case condition $r \ge D_1$ was used.

Step 2 (Case 2: $r < D_1$ and $X_1 = \{v^\star\}$). Here $x = f(\{v^\star\})$, $z = f(R \mid \{v^\star\})$, $d_1 = p$, and we write

$$\delta := \frac{r}{p} > 0. \tag{40}$$

Note $x = \rho_1 p > 0$ because $\rho_1 > 0$ (Step 0). Since $r < D_1$, the first branch of Eq. (19) evaluates $\varphi(r)$:

$$\varphi(r) = x\big(1 - e^{-r/p}\big) = x\big(1 - e^{-\delta}\big). \tag{41}$$

Suppose first $z > 0$. Then $\rho_2 = z/r > 0$ and the case condition $r < D_1 = p \ln \frac{x/p}{z/r}$ can be divided by $p > 0$ to read

$$\delta \;<\; \ln\Big(\frac{x}{p} \cdot \frac{r}{z}\Big) \;=\; \ln\Big(\frac{x}{z}\,\delta\Big).$$

Exponentiating gives $e^\delta < \frac{x}{z}\delta$, i.e.,

$$x \;>\; z\,\frac{e^\delta}{\delta}. \tag{42}$$

Combining this bound with $C \ge 2x$ from Eq. (33) gives

$$\begin{aligned} C + \varphi(r) &= C + x\big(1 - e^{-\delta}\big) \\ &= \tfrac{2}{3}(C + x) + \Big[\tfrac{1}{3}(C + x) - x e^{-\delta}\Big] \\ &\ge \tfrac{2}{3}(C + x) + x\big(1 - e^{-\delta}\big) \\ &> \tfrac{2}{3}(C + x) + z\,\frac{e^\delta - 1}{\delta} \\ &\ge \tfrac{2}{3}(C + x) + z \\ &\ge \tfrac{2}{3}(C + x + z) = \tfrac{2}{3}H \;>\; \alpha H. \end{aligned} \tag{43}$$

Here $C \ge 2x$ gives $(C + x)/3 \ge x$, and Eq. (42) bounds the remaining term $x(1 - e^{-\delta})$ below by $z(e^\delta - 1)/\delta$. The subsequent inequalities use $e^\delta \ge 1 + \delta$, $H = C + x + z$, and $2/3 > \alpha$.

If $z = 0$, then $H = C + x$. Using $C \ge 2x$ and $\varphi(r) \ge 0$ directly gives

$$C + \varphi(r) \;\ge\; C \;\ge\; \tfrac{2}{3}(C + x) \;=\; \tfrac{2}{3}H \;>\; \alpha H.$$

Step 3 (Case 3: $r < D_1$ and $X_1 = R$). Here $z = f(R)$, $x = f(v^\star \mid R)$, and $d_1 = \ell(R) = r$. Since $r < D_1$, the first branch of Eq. (19) evaluates $\varphi(r)$, and because the branch length scale is $d_1 = r$,

$$\varphi(r) = f(X_1)\big(1 - e^{-r/d_1}\big) = z\big(1 - e^{-1}\big) = \alpha z. \tag{44}$$

From Eq. (33), $C \geq 2x$, hence $3C \geq 2C + 2x$, i.e.,

$$C \;\geq\; \tfrac{2}{3}(C + x) \;\geq\; \alpha(C + x), \tag{45}$$

where the second inequality again uses $\frac{2}{3} > \alpha$ (from $e < 3$) and $C + x \geq 0$. Adding Eq. (44) and Eq. (45) and using Eq. (32),

$$C + \varphi(r) \;\geq\; \alpha(C + x) + \alpha z \;=\; \alpha(C + x + z) \;=\; \alpha H.$$

Step 4. The reduction $f(P) = 0$ was handled in Step 0, so assume $f(P) > 0$. If $D_1 \leq r$ — which includes $D_1 = 0$, since $r > 0$ by Eq. (22) — we are in Case 1. If $r < D_1$ — which includes $D_1 = +\infty$, i.e., $\rho_2 = 0$ — we are in Case 2 or Case 3 according to whether the block order puts $\{v^\star\}$ or R first; these two options are exhaustive because (X_1, X_2) is a permutation of $\big(\{v^\star\}, R\big)$. In every case $\widehat{G}(J) + \varphi(r) \geq \alpha \widehat{G}(T)$, which is the claim of Lemma 4. □

B Experimental Details

B.1 Executor and execution protocol

We use Qwen3-32B as the frozen executor throughout all experiments, served with vLLM in FP8. Each rollout consists of a single chat completion at temperature 0.7, with thinking disabled and an output limit of 4,096 tokens. The executor has no tools and is not shown the tests. We adapt each task's BigCodeBench prompt to refer to the private modules and prepend the selected skill documents to the user message in a fixed order, separated by horizontal rules. We tokenize each skill document with the Qwen3 tokenizer and define the token cost of a selected set as the sum of its document lengths, excluding separators and the task prompt.

A rollout succeeds if the code extracted from the completion passes the task's full test suite in the BigCodeBench evaluation harness, run in an isolated process with a 240-second time limit. For each task–set pair, the success rate is the fraction of rollouts that pass all tests. The evaluation process loads the private modules from compiled bytecode with docstrings removed; their source files are not exposed to the generated program.

Offline data collection starts by enumerating every non-empty subset of each task-specific candidate skill pool, with four rollouts per distinct task–set pair. Selected test-task pairs receive one or two additional batches of 12 rollouts, giving 16 or 28 rollouts in total. Subsequent collections for newly added tasks and additional skill combinations use eight rollouts per pair. Pooling outcomes by task–set pair yields 5,204 pairs and 45,400 rollouts for the fitting and evaluation protocols in Appendices B.4 and B.5.

During online training (Appendix B.7), the learner receives one execution outcome for each selected task–set pair. It uses cached outcomes without replacement and collects additional rollouts when needed.

For the final end-to-end evaluation, we collect 16 fresh rollouts for each distinct task–set pair selected by any method. These outcomes are used only for evaluation, and Fig. 5 reports results from this evaluation.

In total, the reported results rest on 90,588 distinct rollouts. They comprise the 45,400 offline rollouts above, the 100 no-skill rollouts of Appendix B.2, 43,088 further rollouts that supply the online-training feedback of Appendix B.7, 20,228 collected before training and 22,860 during it, and the 2,000 rollouts of the final end-to-end evaluation.

B.2 Testbed

We replace `base64`, `binascii`, `hashlib`, `hmac`, and `zlib`/`gzip` with five private modules: `pk64`, `pkbin`, `pkdigest`, `pkmac`, and `pkzip`, respectively. For each operation, we provide

three function variants with different encodings or constants. We randomly designate one variant as canonical and keep this assignment fixed across the tasks, tests, and skill documents. The canonical variants define the behavior expected by the task tests. The library includes documentation of these variants and distractor documents describing non-canonical variants of the same operations.

A modified task passes our screening if its reference solution using the private modules passes all tests, while the original standard-library solution and every hybrid solution fail. Each hybrid replaces the calls to one required private module with the corresponding standard-library operations, while retaining the other private-module calls. Of the 24 tasks that pass this screening, we exclude two with prompt ambiguities unrelated to the private modules and one in which a required private-module operation can be reproduced without its documentation. The final testbed contains 21 tasks: 20 require two capabilities and one requires a single capability. Without injected skills, the executor records 0 successes in 100 rollouts on the modified tasks, compared with 58 successes in 68 rollouts on the original BigCodeBench tasks under the same execution protocol.

The skill library contains 31 documents: 17 cover a single module, 10 cover pairs of modules, and 4 are distractors. Each of `pk64`, `pkbin`, `pkdigest`, and `pkzip` has four documents—a cheat sheet, a quick reference card, a guide, and an FAQ—while `pkmac` has one reference card. Document lengths range from 206 to 1,302 tokens under the tokenizer specified in Appendix B.1.

For the value-model experiments in Section 5.2, we divide the 21 tasks into 11 training tasks and 10 test tasks. The end-to-end experiment uses eight of these test tasks, with 24 instances in Section 5.3. The other two, `00505` and `00934`, are among the four tasks added so that every module's supply is identifiable in Fig. 2: `pkmac`, and `pkzip` apart from `pk64`, are demanded only by them. The optimization instances of Appendix B.6 are assembled from all 21. The remaining 13 tasks are used to warm up the neural capability encoder and the supervised retriever (Appendix B.7). The example in Fig. 1 uses task `00028`, which requires `pk64` and `pkzip`.

B.3 Value-model parameterization

For the value-model experiments in Section 5.2, we instantiate the structured objective of Section 3 with $d = 5$ latent capability dimensions, matching the number of private-module families in the testbed. The task set and skill library are fixed, so the capability encoders are lookup tables: one supply vector $\widehat{\boldsymbol{u}}_i \in \mathbb{R}^5_+$ per skill and one demand vector $\widehat{\boldsymbol{w}}^q \in \mathbb{R}^5_+$ per task. These vectors estimate the effective supplies and demands in Eq. (6), which already absorb the executor-specific factors λ^E_k and η^E_k. We learn the vectors from execution outcomes, without using task or document text or capability annotations. No latent dimension is assigned to a particular module during fitting; the dimensions are matched to modules only when evaluating parameter recovery in Fig. 2.

The fitted objective in Eq. (7) then takes the form

$$\widehat{F}_E(q, S) = \sum_{k=1}^{5} \widehat{w}^q_k \, h_k\left(\sum_{i \in S} \widehat{u}_{i,k}\right) - \widehat{\kappa}_E \ell(S). \tag{46}$$

The executor E and the document lengths ℓ_i are fixed as described in Appendix B.1. We learn the context sensitivity $\widehat{\kappa}_E \geq 0$ jointly with the supply and demand vectors, whose entries are kept nonnegative through a softplus.

We use the same fixed response function in every dimension:

$$h_k(x) = \log\left(\epsilon_0 + (1 - \epsilon_0)(1 - e^{-x})\right) - \log \epsilon_0, \qquad x \geq 0, \qquad \epsilon_0 = 0.01. \tag{47}$$

This function is increasing and concave, with $h_k(0) = 0$ and an upper bound of $\log(1/\epsilon_0)$. It assigns diminishing gains as supply accumulates in a capability dimension and satisfies the conditions in Section 3.1. Neither its functional form nor ϵ_0 is learned.

Since $\widehat{F}_E(q, \varnothing) = 0$, predicting an absolute execution score also requires a task-specific offset $\widehat{b}_q$. We parameterize it as

$$\widehat{b}_q = -a_q - \sum_{k=1}^{5} \widehat{w}^q_k \log(1/\epsilon_0), \qquad a_q \geq 0. \tag{48}$$

This adds one learnable scalar a_q per task; $\widehat{b}_q$ is determined by that scalar and the task's demand vector. The bound on h_k ensures that $\widehat{b}_q + \widehat{F}_E(q, S) \leq 0$, allowing us to convert this score into a probability

as described next. With 31 skills and 21 tasks, the model has $31 \times 5 + 21 \times 5 + 21 + 1 = 282$ learnable parameters.

B.4 Execution success and model fitting

From success probability to execution score. Section 3 allows different execution scores; here we consider task success probability. Let $p_E(q, S)$ be the probability that the frozen executor passes all tests for task q when given the skills in S, under the execution protocol of Appendix B.1. Our goal is to select a feasible set with the highest success probability.

When performance is measured by task success probability, a task's requirements must be considered jointly. In our testbed, a task may use several private-module APIs, and failure in any required operation can cause the entire execution to fail. This motivates modeling success probability as a product of factors associated with the required capabilities. Taking the logarithm turns this product into a sum of capability contributions, consistent with the demand-weighted benefit in Section 3.1. We therefore define the execution score on a log-probability scale in these experiments.

To keep the score finite even when the success probability is zero, we first apply the same smoothing used for the execution labels:

$$p_{E,\epsilon}(q, S) = \epsilon_0 + (1 - \epsilon_0)p_E(q, S), \qquad J_E(q, S) = \log p_{E,\epsilon}(q, S). \tag{49}$$

Here J_E is the execution score introduced in Section 3, and $\epsilon_0 = 0.01$ is the fixed constant in Eq. (47). Both smoothing and the logarithm are strictly increasing, so for each task,

$$\arg\max_{S \in \mathcal{F}_B} p_E(q, S) = \arg\max_{S \in \mathcal{F}_B} J_E(q, S) = \arg\max_{S \in \mathcal{F}_B} F^\star_E(q, S). \tag{50}$$

The last equality follows from $F^\star_E(q, S) = J_E(q, S) - J_E(q, \varnothing)$. Thus, the score transformation preserves the original selection objective.

Predicting success and fitting the parameters. The structured objective predicts a score improvement. Adding the task offset estimates the absolute score, giving the smoothed probability prediction

$$\widehat{p}_{E,\epsilon}(q, S) = \exp\Big(\widehat{b}_q + \widehat{F}_E(q, S)\Big). \tag{51}$$

The offset parameterization in Eq. (48) ensures that this prediction never exceeds one. The offset $\widehat{b}_q$ represents the model's no-skill score because $\log \widehat{p}_{E,\epsilon}(q, \varnothing) = \widehat{b}_q$. For a fixed task, the offset is constant across sets, so maximizing $\widehat{F}_E$ also maximizes the predicted probability.

Each rollout provides an outcome $y \in \{0, 1\}$ indicating whether all tests passed. We smooth it to $t = \epsilon_0 + (1 - \epsilon_0)y$ and fit the model by minimizing binary cross-entropy,

$$\mathcal{L}(t, \widehat{p}) = -t \log \widehat{p} - (1 - t) \log(1 - \widehat{p}), \qquad \widehat{p} = \widehat{p}_{E,\epsilon}(q, S), \tag{52}$$

averaged over the training rollouts. The conditional mean of t is $p_{E,\epsilon}(q, S)$, so the smoothed labels and probability prediction use the same target. We optimize the parameters jointly with full-batch Adam for $4{,}000$ steps, using an initial learning rate of 0.05 and a cosine schedule. We run three random restarts and retain the fit with the lowest training loss. The offline data contain the $5{,}204$ measured task–set pairs of Appendix B.1, with pooled success and failure counts; the protocols in Appendix B.5 specify which records are used for fitting. The no-skill executions are excluded from fitting and are used only to evaluate the empty-set choice.

B.5 Value-model validity

We provide implementation and evaluation details for Section 5.2 and supplement its prediction results with two experiments on generalization to held-out skill combinations.

Comparison models. For the value-model comparison in Section 5.2, every model receives the task identity and a binary membership vector over the 31 documents; none uses text. Additive fit learns one coefficient per task and document plus a task offset (672 parameters). Shapley scores use the same additive form, with each document's value set to its mean marginal contribution over all training pairs $(S, S \cup \{i\})$ (651). Datamodels fit one ridge regression per task to predict the smoothed success

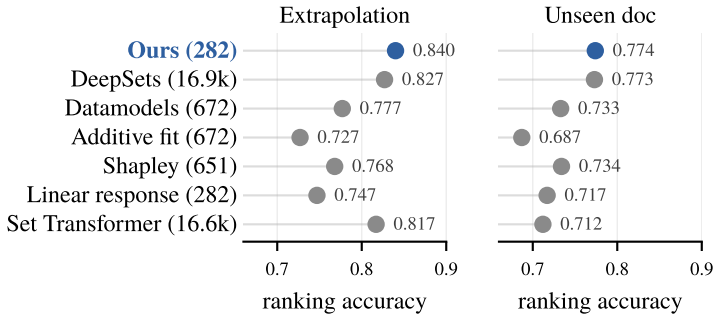

Figure 6: Supplementary prediction experiments for Section 5.2, reporting pairwise ranking accuracy with parameter counts in parentheses. *Left (extrapolation):* training uses sets of one or two documents; testing uses sets of three or more. *Right (unseen doc):* each fold withholds all combinations containing one target document, retains its singleton records in training, and tests on the withheld combinations.

rate from the membership vector, with the ridge strength chosen on a held-out fifth of that task's training sets (672). DeepSets embeds each document in 32 dimensions, sums the embedded members through an MLP, and concatenates a learned task embedding before the output MLP (16,929); Set Transformer replaces the sum with one four-head self-attention block and attention pooling (16,577). Linear response replaces the saturating function in our model with $h_k(x) = x$. The neural models use the same smoothed binary cross-entropy, restarts, and step budget as our model in Appendix B.4, with weight decay 10^{-4}; Datamodels and Shapley use least squares and averaged marginal contributions, respectively.

Main prediction experiment. Ranking accuracy measures how often a model correctly orders two sets from the same task whose measured success rates differ by more than 0.25; a predicted tie receives half credit. For Fig. 4, every model is fitted on all measured sets of the 11 training tasks and on 20 sets of each test task, sampled once with a fixed seed. Evaluation uses the remaining 2,806 sets of the test tasks. Linear response predicts a rate of one on every held-out set, so all of its comparisons count as ties.

Main selection experiment. For the selection panel of Fig. 4, we randomly split each task–set pair's rollouts into two halves, fit every model on one half, and evaluate its selected sets on the other. Fitting uses every measured set of the 11 training tasks and 20 measured sets of each test task. The 204 selection instances use candidate pools of four or five documents over eight test tasks, with budgets from 689 to 1,593 tokens. Each model enumerates every non-empty feasible subset and selects the one it scores highest. The empirical ceiling averages the highest measured success rate among these subsets in each instance, using the evaluation half. We repeat the split with three random seeds and report the mean success rate. On every seed, the sets our model selects succeed more often than those of DeepSets and Set Transformer, by 0.06–0.12 and 0.03–0.10 respectively.

Supplementary prediction experiments. The main prediction experiment evaluates held-out sets after fitting on a sample of sets from each test task. The two experiments in Fig. 6 impose specific restrictions on the combinations available during fitting: one withholds all larger sets, and the other withholds all combinations containing a particular document. Each protocol repartitions the full 5,204 measured task–set pairs by set composition. All models use the same partitions within each protocol and are evaluated by the pairwise ranking accuracy defined above. Both experiments examine generalization to new combinations within the fixed task set.

Extrapolation: from small sets to larger combinations. This experiment asks whether a model can predict the relative value of larger skill sets after learning only from singletons and pairs. We use a single split: the training set contains the 2,099 measured task–set pairs with one or two documents, and the test set contains the remaining 3,105 pairs with three or more documents. No execution outcomes from the larger sets are used for fitting. Our model achieves a ranking accuracy of 0.840, compared with 0.827 for DeepSets (Fig. 6, left). This result supports the fitted objective's ability to rank skill sets beyond the sizes observed during training on this testbed.

Unseen doc: predicting new combinations of an observed skill. This experiment asks whether a model can rank combinations containing a document when that document has been observed only on its own. We use 17 folds, one for each single-module document. In a given fold, the training set contains all measured task–set pairs whose set excludes the target document, together with all pairs whose set consists of that document alone. The test set contains all measured pairs in which the target document appears with at least one other document. Thus, its singleton outcomes remain available for fitting, while every combination containing it is held out. Our model achieves a ranking accuracy of 0.774, close to DeepSets at 0.773 (Fig. 6, right) with 60× fewer parameters. The result supports predicting new combinations of a known skill from its singleton observations and the observed combinations of other skills.

Sensitivity to smoothing. With $\epsilon_0 = 0.05$ in place of 0.01, the share of instances on which BPS reaches the reference in Appendix B.6 changes by at most 0.013 at any library size, and BPS remains the top method at every size.

B.6 Optimization quality

We evaluate how well BPS solves the fitted skill-selection problem in Eq. (8) as the library grows. The instances are built from the fitted parameters, with libraries of 31 to 124 skills and selected sets of up to twenty. Within each instance, all methods use the same objective and token budget, and we evaluate their selected sets under that objective. This comparison isolates optimization quality from the model's ability to predict execution success.

Libraries and instances. We build libraries larger than the testbed's by tiling its fitted parameters. The parameters come from the structured model of Appendix B.3 fitted on all 5,204 measured task–set pairs, the fit shown in Fig. 2, shifted so that ∅ scores zero: a supply vector and a length for each of the 31 skills and a demand vector for each of the 21 tasks, all over five capability dimensions. Figure 7 shows a library with three families: each family is a copy of the fitted library with its own five capability dimensions, its supplies and lengths perturbed by log-normal noise with scale 0.2 and 0.1. A task is assembled from one fitted task per family, drawn at random from the 21. It places each drawn task's demand on that family's dimensions, so covering it takes skills from every family. We take one, two and four families, giving $L = 31$, 62 and 124 skills, and assemble 20 tasks for each library size. For each task we find the cheapest set of skills that covers its capabilities and set the token budget B to 1, 1.5, 2, 3 and 5 times that cost. The instances add to the objective a per-token price ρ/B with $\rho \in \{0, 1, 2\}$, the settings of Section 5.3, giving 300 instances per library size. The sets BPS selects hold two skills on average at $L = 31$ and twelve at $L = 124$.

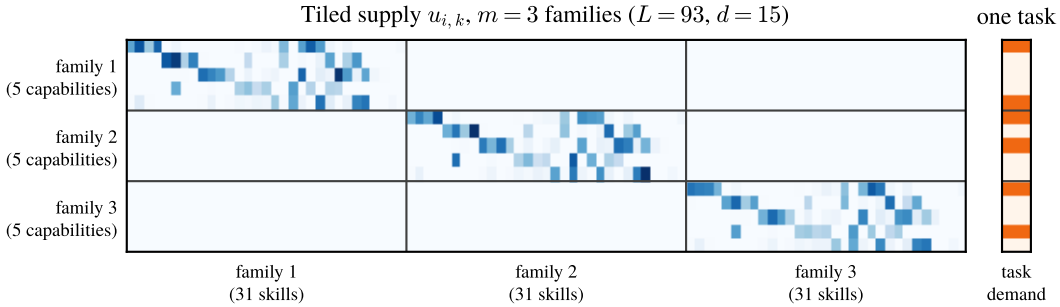

Figure 7: A tiled library with three families. *Left:* the supply matrix, 15 capabilities by 93 skills; each family is a perturbed copy of the fitted library in Fig. 2 over its own five capability dimensions. *Right:* one task's demand, one fitted task per family.

Baselines. For a fixed instance, write $f(S)$ for its fitted objective, including the added per-token price. Every method respects the token budget.

- **Density greedy** starts from the empty set and repeatedly adds the feasible skill maximizing $[f(S \cup \{i\}) - f(S)]/\ell_i$ until no further skill fits. It returns the better of its final set and ∅ under f, without retaining intermediate prefixes.
- **Gear** (Guo et al., 2026) follows the density rule of cost-benefit greedy (Khuller et al., 1999) and stops once no remaining skill increases f. It returns the better of this set and the best single skill.

- **Greedy+Max** (Yaroslavtsev et al., 2020) follows the density-greedy chain and, at each step, also forms the candidate that augments the current set with the feasible skill of largest marginal gain $f(S \cup \{i\}) - f(S)$. It returns the candidate with the highest value of f. The original analysis assumes a monotone objective, so we stop the chain once no feasible skill increases f.
- **Twin Greedy** (Gong et al., 2024) grows two disjoint sets in parallel, each time adding the skill and set with the largest ratio of unpriced marginal gain, less twice the price of the skill, to that price. It restarts from every feasible seed of at most two skills and returns the best set found.
- **Threshold Twin Greedy** (Gong et al., 2024) is its faster variant, which replaces each exact maximization with a decreasing threshold.
- **Best-of-200 random** generates 200 random feasible candidates, each filled until no further skill fits, and returns the candidate with the highest value of f.

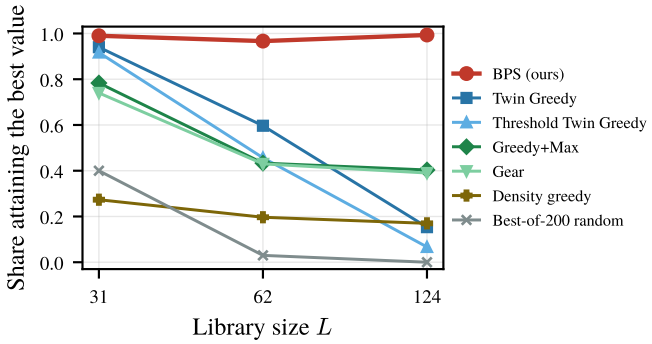

Figure 8: Optimization quality as the library grows. Each point is the share of instances on which a method's selected set reaches the best value found by any method, over 300 instances per library size.

Results. Enumeration is out of reach at these sizes, so on each instance we take the highest objective value any compared method finds as the reference, and a method attains it when its selected set's value is within 10^{-9} of that value. The reference is close to the optimum: relaxing the selection to fractional skills gives a concave problem whose Frank–Wolfe solution upper-bounds the optimum, and this bound exceeds the reference by only 0.5% on average at $L = 31$ and 0.1% at $L = 124$. Figure 8 reports how often each method attains the reference. `BPS` does so on 99%, 97% and 99% of the instances at $L = 31$, 62 and 124; on the remaining 16 instances Twin Greedy finds a set whose value is higher by less than 0.02%. Averaged over the three sizes, `BPS` attains the reference on 98% of the instances and Twin Greedy on 56%, as quoted in Sections 1 and 4. Twin Greedy, the strongest baseline at $L = 31$ with 94%, falls to 15% at $L = 124$ at a comparable number of objective evaluations, and Threshold Twin Greedy falls from 92% to 7%. Greedy+Max and Gear fall from 78% and 74% to 40% and 39%, density greedy stays below 30%, and best-of-200 random drops from 40% to none. We also remove one part of `BPS` at a time. A single density chain that keeps all its prefixes attains the reference on 39% of the instances at $L = 124$, and the enumerated seeds that keep only the endpoint of each chain on 49%. End-to-end execution success with the neural capability encoder is evaluated in Section 5.3.

B.7 End-to-end selection

The neural encoder embeds the task prompt and each document with frozen `text-embedding-v4` and applies two linear maps into $d = 64$ latent capabilities; supplies are a softplus of one map and demands a softmax of the other. It uses the fitted objective in Eq. (7) with the response function in Eq. (47).

Warm-up uses no executor. From the capability annotations of the other 13 tasks, the 11 training tasks of Appendix B.2 and the two test tasks not evaluated here, we enumerate the minimal covering sets and train the maps for 400 Adam steps at learning rate 5×10^{-4} to score each cover above the sets that drop one of its documents or add one, and above the other sets of its task. Online training then runs 17 passes over the other tasks' instances at budgets ranging from 500 to 2,000 tokens, with the added price of Section 5.3 set to $\rho = 0$. Each step decodes a set with `BPS` under Thompson sampling

over an ensemble of ten encoders, observes one pass/fail rollout of that set, and updates the maps on the smoothed log loss at learning rate 10^{-3}.

The evaluation is leave-one-task-out: for each of the eight test tasks we remove its instances from the stream, train, and decode its instances at the three budgets and three prices of Section 5.3, so the reported selections come from encoders that never received feedback on the task they select for. We repeat this with three training seeds and average each learned selector over them. On these neural objectives `BPS` attains the exact optimum: enumerating every feasible subset returns its set on all 216 combinations of instance, seed and price.

We execute every set chosen by any rule on any instance, 125 distinct task–set pairs, with 16 rollouts each. A rule's success is its sets' pass rate averaged over the 24 instances, and its tokens are the summed document lengths of its sets averaged the same way. Table 1 lists every point of Fig. 5.

Table 1: Every point of Fig. 5: measured success and injected tokens per budget, and their mean over the three budgets. Each cell reads success / tokens; the self-selecting executor's tokens include the catalogue lines it reads.

Rule	$B = 500$	$B = 750$	$B = 1{,}000$	Mean
`BPS`, $\rho = 0$	0.66 / 464	0.65 / 686	0.70 / 927	0.67 / 693
`BPS`, $\rho = 1$	0.62 / 456	0.66 / 513	0.66 / 550	0.65 / 506
`BPS`, $\rho = 2$	0.49 / 370	0.52 / 410	0.62 / 456	0.55 / 412
Executor self-select	0.51 / 530	0.57 / 660	0.71 / 753	0.60 / 647
Fine-tuned retriever, $k = 1$	0.00 / 222	0.00 / 222	0.04 / 302	0.01 / 249
Fine-tuned retriever, $k = 2$	0.66 / 464	0.48 / 523	0.44 / 565	0.53 / 518
Fine-tuned retriever, $k = 3$	0.66 / 464	0.48 / 686	0.64 / 909	0.60 / 687
SkillRouter, $k = 1$	0.00 / 373	0.00 / 520	0.00 / 520	0.00 / 471
SkillRouter, $k = 2$	0.26 / 453	0.12 / 604	0.27 / 822	0.22 / 626
SkillRouter, $k = 3$	0.26 / 453	0.12 / 604	0.27 / 878	0.22 / 645
Graph-of-Skills	0.12 / 354	0.06 / 569	0.00 / 846	0.06 / 590
BM25	0.00 / 471	0.12 / 691	0.12 / 934	0.08 / 699

Read on the same rounds, the lookup tables of Section 5.2, fit on the test tasks, reach 0.70 success on 695 tokens. They mark the fitted objective's ceiling on these instances rather than a selector that a deployment could run.

The deployed systems rank the library once per task. BM25 runs over the full document bodies with the task prompt as query. SkillRouter runs the authors' code with their released 0.6B embedding and reranker models, the reranker reordering the embedding's top 20. Graph-of-Skills runs the authors' code on a graph built from our 31 documents and fills our token budget in its ranking's order, skipping documents that do not fit. The self-selecting executor is shown the task, the 31 catalogue lines and the budget and returns a list, truncated to the budget in its order. Budgeted top-k walks a ranking from the top, skips any document that would exceed the budget, and stops after k documents; both curves of Fig. 5 saturate by $k = 3$.

The fine-tuned retriever scores each document alone with a bilinear head on the same frozen `text-embedding-v4` embeddings, two linear maps into 64 dimensions with a scale and an offset. Warm-up trains it for 400 Adam steps at learning rate 5×10^{-4} on the covering sets of the same 13 training tasks, labelling the documents of a cover positive and the rest of the library negative. It then runs its own online phase on the same stream and the same 17 passes as the encoder, one rollout per step at learning rate 10^{-3}: it fills the budget in rank order, executes the set, and updates every document in it toward the set's pass/fail outcome under the smoothed log loss. The evaluation is the same leave-one-task-out protocol, and its selections are read at budgeted top-k.

C Limitations and Future Directions

C.1 Nonlinear context penalties

The penalty $c_E(S) = \kappa_E \ell(S)$ in Section 3.1 assigns the same marginal cost to every injected token. This is a first-order approximation of score degradation, expressed in the same units as the capability

benefit. A nonlinear penalty $g_E(\ell(S))$, with $g_E(0) = 0$ and g_E nondecreasing, would allow the marginal cost to vary with the total injected length. For example, a convex penalty could represent degradation that becomes steeper as context grows, while a saturating penalty could represent a diminishing effect of additional length. Which form is appropriate for a given executor remains an empirical question.

Evaluating these alternatives would require separating length effects from changes in useful capability information. One approach would hold the required API information fixed while varying the amount of redundant or unrelated text, then compare linear and nonlinear penalties on held-out execution records. The evaluation should measure both prediction quality and the execution success of the selected sets under the same token budgets. Better prediction alone would not establish that a more flexible penalty improves selection. On the theoretical side, Theorem 1 relies on a penalty proportional to token length. Extending the guarantee requires identifying nonlinear penalty families for which efficient selection algorithms can still provide a useful bound on benefit and cost.

C.2 Interactions beyond capability coverage

In Eq. (3), each skill has a fixed capability supply vector. We add supplies within each dimension and apply h_k, then weight coverage by task demand. The nonlinear response makes each skill's marginal benefit depend on the selected set, capturing redundancy and complementarity. Additional dimensions can represent knowledge needed to combine capabilities, such as how to connect two APIs.

A limitation is that this representation does not explicitly model conflicts or confusion between particular skills. Two useful documents may prescribe incompatible conventions or describe easily confused operations, so their joint injection could reduce execution success even when capability coverage is sufficient.

We make this simplification because we assume a sufficiently capable executor that can distinguish different skills' roles and resolve common ambiguities from their documentation. Under this assumption, residual adverse interactions are limited enough for the coverage model to be useful. Its adequacy depends on the executor and the task.

Future work could model these residual effects through interaction terms learned from execution feedback, then study how to optimize the extended objective under a token budget. Because such terms may violate diminishing returns, obtaining provable guarantees requires identifying suitable restrictions on the interactions or developing new selection algorithms.