
BehaviorGPT-v4: One Large Behavioral Model for Retail, Engagement, and Payments with Zero-Shot Transferability

Rickard Br��l-Gabrielsson [†]	Vasudev Gupta	Tom Boustedt [*]	Adam Fredriksson [*]	
Simon Granstr��m [*]	Marcel R��d	Gon��alo Marques	Jens Palmborg	Erik Guander
Nicolas Sanchez	Artem Lukoianov	Srijan Sood [‡]	Georgios Kolovos [‡]	Simon Ejdemyr [‡]
Toni Rosino [‡]	Mattias Holmstr��m [‡]	Marc Mathieu	Pratik Thakar [‡]	Wen Yao [‡]
Miguel Paredes	Peter Sarlin [‡]	Alexander Statnikov [‡]	John Br��l-Gabrielsson	
Steve Flinter [‡]	Bryan McCann [‡]	Sebastian Siemiatkowski [‡]	Gunnar Carlsson	

Unbox AI

Abstract

Language modeling had two defining moments. First, a single pretrained model, fine-tuned per task, displaced task-specific systems. Then scale made fine-tuning optional for many tasks, letting the same model handle them from few or no examples. Behavioral models learn from sequences of human actions, such as purchases, payments, and app sessions, but remain organized around individual datasets, tasks, and domains. We present BehaviorGPT-v4, a 12.5B-parameter Large Behavioral Model pretrained by next-event prediction on 150 billion user actions (3.6 trillion tokens) from retail, engagement, and payments. It represents behavior as sequences of discrete, multimodal events and extends next-token prediction to an open set of candidates, still scored in a single step. On public datasets whose catalogs and users were held out from pretraining, BehaviorGPT-v4 with no target-specific training outperforms every sequential, generative, transferable, tabular, and LLM baseline we evaluated, including specialists trained on the target’s full training split. Fine-tuned, it learns from added target data 17 times faster than the strongest baseline. The same model serves three industries and 13 datasets with no per-dataset changes: it leads the strongest baseline on 30 of 31 dataset–task pairs without adaptation and on all 31 fine-tuned. The 12.5B model transfers better zero-shot than our 0.5B model and needs less target data. It ranks a full catalog in one forward pass, in under a millisecond per query, 48 times faster than a prompted LLM. Production A/B tests of the BehaviorGPT family show double-digit gains in conversion and orders in several deployments. We document the recipe and design decisions behind the BehaviorGPT family, with internal ablations. An SDK and API for zero-shot deployment and evaluation on new catalogs, and live demos built on them, are available at www.unboxai.com/behaviorgpt.

Technical report, October 8, 2026. [†]Project lead. ^{*}Equal contribution, listed alphabetically. [‡]Not affiliated with Unbox AI; contributing in a personal capacity. Author contributions [listed at end of paper](#). Correspondence: rickard@unboxai.com

Contents

1	Introduction	3
2	Approach	10
2.1	Desiderata	10
2.2	Behavior as Sequences of Events	10
2.3	Events as Multimodal Features	11
2.4	Model, Scale, and Training Configuration	12
2.5	Losses and Candidate Sampling	13
2.6	Data	13
2.7	Fine-tuning	14
2.8	Inference	14
2.9	Benchmarks and Evaluation Axes	14
3	Results	16
3.1	Public Datasets	16
3.2	Breadth: One Pretrained Model Across Industries and Tasks	16
3.3	Scaling Laws	19
3.4	Data Mixing	21
3.5	Internal Experiments and Ablations	21
3.6	Real-World Deployments (A/B Tests)	22
3.7	Qualitative Examples	22
4	Demos, SDK, and API	23
5	Discussion	24
5.1	Limitations	24
5.2	Ethics and Societal Impact	25
6	Related Work	26
6.1	Foundation Models	26
6.2	Recommendation Systems	27
6.3	Large Behavior Models in Robotics	27
6.4	Behavioral Foundation Models	28
6.5	LLMs, Agents, and Reasoning for Behavior	29
7	Conclusion	30
A	Extended Definition and Positioning	41
B	Architecture and Objectives	44
C	Data	51
D	Training, Systems, and Metrics	54
E	Public Datasets in Detail	57
F	The Breadth Matrices in Detail	59
G	Internal Experiments and Ablations	77
H	A/B Test Details	83
I	Qualitative Examples from the Demos	88
J	Applications	98

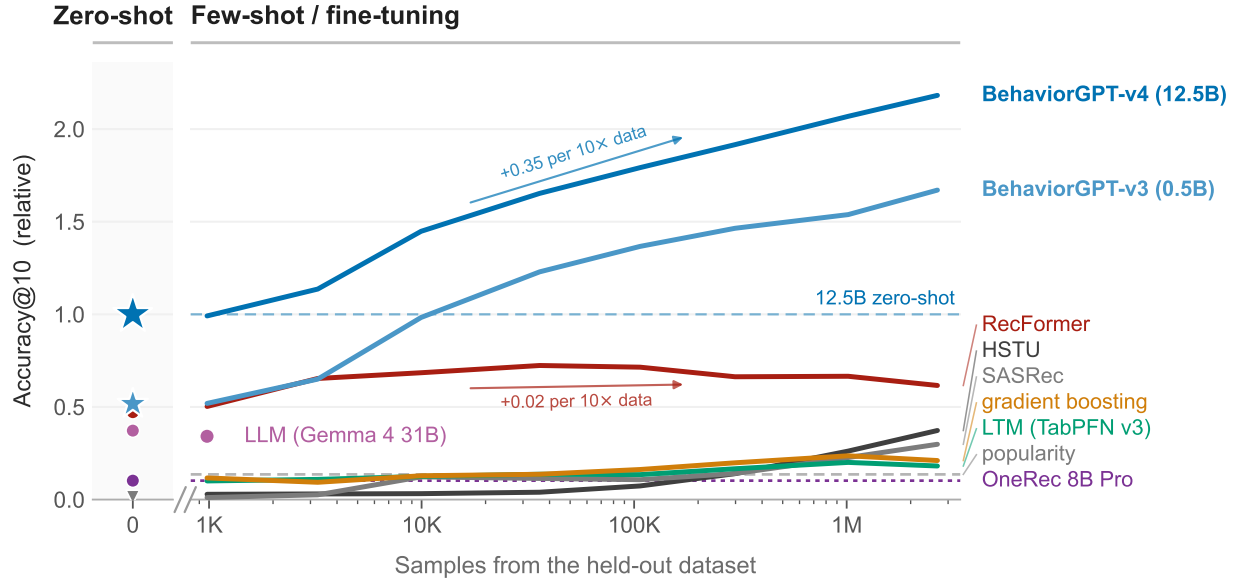


Figure 1: **Large Behavioral Models transfer zero-shot to datasets held out from pretraining, and stay ahead of baselines given millions of target samples.** Two held-out datasets averaged into one panel, short video (PixelRec) and reviews (Amazon Sports). Accuracy@10 is Effective Recall@10 under full-catalog ranking on a temporal split, with unseen items and users, divided by the 12.5B’s own zero-shot score on the same dataset (0.594% and 0.763%), so the dashed rule at 1.0 is our zero-shot point and every value is a multiple of it. *Left*: no samples from the target, though the LLM’s point ranks a shortlist of the target’s most frequent products. *Right*: accuracy after adaptation on N target samples, by fine-tuning for ours, training for the specialists, a refit over the budget’s own samples for gradient boosting (LightGBM), and in-context conditioning for the LLM and for the LTM (TabPFN v3). No baseline reaches the rule at any budget. Arrows are accuracy gained per tenfold increase in target data, fitted over the sweep: +0.35 for ours against +0.02 for RecFormer, the strongest baseline. See Appendix F.4 for details.

1 Introduction

Foundation models have transformed language, vision, and genomics [9]. They have made less headway in several of the industries where data and use cases are most abundant: in retail, engagement (app usage), and payments, most of what is logged is behavioral (records of what people did rather than what they wrote or photographed), and it is still modeled one dataset, one task, and one domain (company, vertical, or market) at a time. We present BehaviorGPT-v4, which extends the paradigm to all three [36, 34, 35]: a *single* 12.5B-parameter model trained on 150 billion real human actions and 3.6 trillion tokens, one pretrained model behind every domain and task in this paper. Both numbers are, to our knowledge, the largest yet *disclosed* for behavior: $1.5\times$ the largest published event count and $1.8\times$ the largest published token count. The same holds for the model: its 12.5B non-embedding parameters are more than ten times the largest reported for any behavioral foundation model (setting aside sizes inherited from language-model backbones and totals whose non-embedding share is undisclosed; Section 6.4 gives the accounting). Our BehaviorGPT models, deployed with retailers and payment providers, have driven double-digit increases in conversion and orders in A/B tests against the incumbent systems, some running for several years (Section 3.6).

Two moments for language. In language, one pretrained model fine-tuned per task first displaced the task-specific architectures that preceded it, on many tasks at once [54, 88, 26, 91]; then scale made the fine-tuning optional [89, 11], the same model handling tasks it had never been trained on, improving at them as it grew, and on some of them matching or outperforming systems trained on those tasks in full. Both results rested on one formulation, next-token prediction, which absorbed tasks that had been modeled apart. Behavior has had neither, and the same unification is missing: the pretrained model is mostly a *feature extractor*, with a head fitted per task. We pose the tasks as one, next-item retrieval, and take it to a scale at which it has not been tried (Section 2). One pretrained model then reproduces both moments: fine-tuned per dataset, it beats every baseline on every task (Figures 2 and 5); left unadapted, the same fixed model makes fine-tuning optional (Figures 1 and 11).

	Retail								Engagement				Payments				
	Marketplace	Grocery 1	Grocery 2	Pharmacy (browsing)	Pharmacy (transactions)	Convenience	Home decor	Camera & photo	Beverage discovery	Image discovery	Short video	Reviews	Merchant	Category	mean rank	median % of leader	
BehaviorGPT 12.5B fine-tuned	100	100	100	100	100	100	100	100	100	100	100	100	100	100	1.0	100.0	
BehaviorGPT 12.5B zero-shot	100	86	82	84	76	46	75	26	91	100	35	63	100	100	2.5	83.1	
BehaviorGPT 0.5B fine-tuned	87	82	87	92	74	82	87	91	40	28	72	83	78	75	2.7	82.3	
BehaviorGPT 0.5B zero-shot	87	75	73	84	64	41	75	23	4.7	12	7.8	51	78	75	5.1	68.8	
RecFormer	26	8.1	12	59	25	36	56	30	11	3.3	25	31	70	54	T	7.3	28.0
HSTU	10	21	19	51	64	46	59	41	2.6	1.3	10	32	45	11	T	7.9	26.3
LLM (Gemma 4 31B)	9.4	29	34	43	35	24	39	31	10	3.1	4.7	38	54	69	B	8.1	32.4
SASRec	12	24	22	53	54	33	63	60	2.8	1.7	7.6	25	13	11	T	8.1	23.4
user-popular	45	7.2	1.7	40	0	40	56	19	13	21	0	0	83	71	w	8.7	20.3
recent items	50	6.7	1.5	40	0	37	57	19	13	21	0	0	85	77	w	8.8	20.1
RecFormer zero-shot	7.2	1.9	1.7	38	19	19	35	12	0.80	3.3	14	33	66	40	w	10.5	16.6
OneRec 8B Pro	8.3	6.6	2.8	39	7.8	26	30	13	0.64	5.1	5.1	3.7	59	38	w	11.1	8.0
popularity	6.7	7.9	6.4	6.2	50	15	13	19	1.1	0.93	1.0	16	61	34	p	11.3	10.6
LTM (TabPFN v3)	2.3	1.2	1.7	3.5	5.3	5.5	6.3	23	1.0	0.68	1.0	21	68	46	B	12.1	4.4
gradient boosting (LightGBM)	0.50	1.1	1.2	1.3	2.9	1.7	2.7	5.6	0.30	0.37	1.1	25	51	8.4	B	14.1	1.5

Figure 2: **Every method on every dataset, as a share of the best score on that dataset.** Recall@10. A cell is the method’s recall as a percentage of the highest any method reached on that dataset, so 100 marks the leader and 10 an arm an order of magnitude behind it. Letters mark what target data each row was given, as Table 9 sets out: **T** trained on the target’s training split, **p** counted over its training stream, **w** no target data at all, **B** a bounded candidate list of the most frequent products in the target’s own data. Our own four checkpoints are set in bold. Rows are ordered by mean rank over all fourteen columns, with the median of the row in the column beside it. The Payments band is scored by hit@5 rather than recall@10 and counts toward both. See Appendix F.8 for details.

Our claim. We claim the same transition is available for behavior, and that three falsifiable properties would show it: (i) **zero-shot transfer**: a pretrained model that ranks well on a new merchant, catalog, and user population with no samples from that dataset, where a specialist has nothing to train on at all; (ii) **breadth from one pretrained model**: one fixed set of weights competitive with the strongest specialist across retail, engagement, and payments, across the subdomains within each of them, and across task families, all simultaneously rather than one at a time, since breadth along one axis alone would leave the per-dataset model as the default; and (iii) **generality that improves with scale**: zero-shot transfer getting better, and adaptation getting more sample-efficient, as model scale grows, which we test by comparing our 0.5B and 12.5B models. Four figures test them: two held-out datasets from zero shot through a full adaptation sweep (Figure 1), every method in the benchmark against every dataset in it on the one slice all of them share (Figure 2), the adapted model against the strongest specialist on every dataset and task we can pair them on (Figure 5), and what model scale bought on that grid, without target data and with it (Figure 12).

The third property was central to GPT-3 [11], whose case rested less on any one benchmark than on performance on untrained tasks rising steadily with scale, and on larger models making better use of in-context examples. We test the behavioral analogue at our two model sizes (Section 3.1): 25× more parameters improve zero-shot transfer on 28 of 31 dataset–task pairs.

Zero-shot for behavior. Zero-shot has never meant a model untouched by a task, only one never trained on it *explicitly*: GPT-3’s held-out tasks were latent in its corpus, so the term marks a position on a spectrum rather than a clean break. Behavior differs in where the novelty lies. Words are shared across every English corpus, so a new corpus can be new only in what it asks. A behavioral dataset’s catalog, by contrast, *is* its vocabulary, so a new dataset is new in its symbols, and reaching an unseen catalog at all requires describing actions by their multimodal features rather than their identities. Tasks must reach the model the same way. A language model is told its task in words; ours is told it in behavior, with every task posed as the next action given the context that defines it (Section 2.2). Finding one such formulation for all tasks is much of what the approach has to solve. Figure 1 is the far end of that spectrum: two public datasets whose catalogs, items, and users are absent from pretraining, in domains for which we hold no data at all. The datasets of Figure 2 are mixed. Several catalogs are new in the same sense; for others, some items were seen in

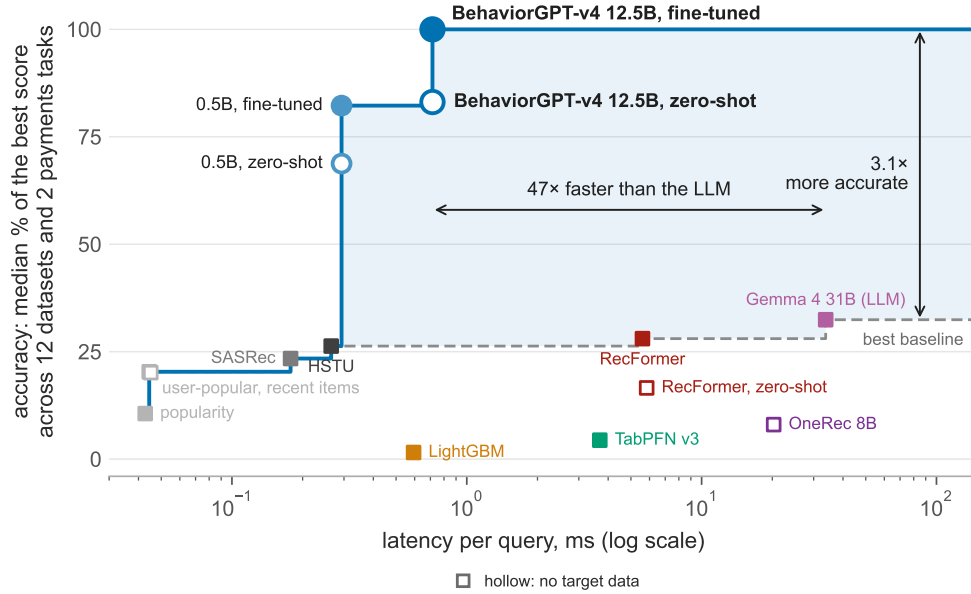


Figure 3: **BehaviorGPT-v4 is the most accurate method on the benchmark at sub-millisecond serving cost, the cost band of the cheapest learned recommenders.** Each point is one method: its median share of the best score on each dataset (Figure 2) against its median milliseconds per query (Table 3). Hollow points use no target data. The shaded band runs between the best baseline at each cost (dashed) and the best method overall (solid). The fine-tuned 12.5B reaches $3.1\times$ the median of the strongest baseline, Gemma 4 31B, at $48\times$ less time per query. The gap is architectural: like SASRec and HSTU, BehaviorGPT ranks the whole catalog in one forward pass, whereas semantic-ID recommenders and prompted LLMs decode in multiple steps. Latencies are harness throughput, not serving latency, and Gemma and the tabular arms are timed on a thousand-item shortlist, a smaller ranking problem than ours (Appendix F.3).

pretraining, and the task belongs to a family pretraining covers, though all users are new throughout. Zero-shot covers a weaker claim there.

Target data and engineering cost. Zero-shot and few-shot are first of all claims about cost: GPT-3’s case rested on the expense of a labeled set per task [11]. Two costs matter here, target data and the engineering around it, and Figure 1 shows both: a training pipeline per dataset for a specialist, a light fine-tune for us, and nothing at the left edge. The few dozen examples that few-shot means in language rarely bind here, since any traffic at all logs thousands of events in a week; we therefore sweep from a thousand target samples up to the full training split. Each tenfold increase in target data adds $+0.35$ of our zero-shot accuracy, against $+0.02$ for the strongest baseline. Even given the entire split, that baseline reaches only $0.62\times$ our zero-shot score, while our fine-tuned model reaches $2.18\times$.

Where prior work stands. Behavioral foundation models have reached industry: over the past two years, media, commerce, banking, card networks, payments, and markets have each published one (HSTU at Meta [112], Shopify’s commerce model [3], PRAGMA at Revolut [83], nuFormer at Nubank [10], and five more in Table 1). Each is pretrained on one company’s event stream and evaluated on that company’s internal tasks, with headline results reported as relative lifts on proprietary data. These models show that scale helps *within* a platform, but almost none tests transfer *across* platforms, domains, and catalogs, the property central to foundation models: where public datasets appear, as in HSTU, models are trained on them from scratch (Section 6.4). Academic work [27, 50, 51, 72, 60] does report zero-shot transfer, but at 10^{-5} – 10^{-4} of our pretraining scale, on one industry’s public recommendation corpora, and almost always for next-item prediction alone. Each line of work is thus broad in one respect and narrow in the others: in tasks but not platforms, or in catalogs but not industries or tasks. A foundation-model claim requires breadth on all of them at once and at industrial scale, which to our knowledge has not been shown. We evaluate on public datasets so that both the cross-domain and zero-shot claims can be verified, and release a notebook that reproduces our zero-shot PixelRec evaluation through the public API. Table 1 summarizes this landscape and Section 6 develops it.



Figure 4: **The language of behavior.** An LLM predicts the next token from a fixed vocabulary; a Large Behavioral Model predicts the next action, where each event can carry text, imagery, a time, a place or a price, and the candidates form an open catalog. Language is the special case in which every event is a single token and the catalog a fixed vocabulary (Section 2.3).

Large Behavioral Models. We define **Large Behavioral Models** (LBMs) as foundation models in which co-occurrence within chronological sequences of human actions defines the meaning of the entities acted upon. As LLMs model language by next-token prediction, we model behavior by next-action prediction over an individual’s chronologically ordered actions (Figure 4); with every event a single token and the catalog a fixed vocabulary, the formulation reduces to an LLM. Conversely, any task whose answers can be enumerated and embedded becomes a single retrieval step rather than autoregressive decoding (Section 2.3). LBMs differ from LLMs and vision-language models (VLMs) in what they observe: text and images are the polished *outputs* of human activity, while behavioral logs record the *trajectory* that produced them. This changes what similarity means. To an LLM or VLM, regular and lactose-free milk are near-duplicates; behaviorally they are nearly disjoint (Figure 6). Models that derive similarity from descriptions produce answers that sound *reasonable*; models that derive it from actions produce answers matched to *what the user at hand actually does*. This principle shapes BehaviorGPT’s embeddings, losses, and candidate sampling, and separates it from zero-shot recommenders whose transfer runs entirely through item text, and so through the description-based similarity this example rejects (Section 6.4). A companion position paper [33] and Appendix A develop the definition and its consequences.

Why one model per domain has been the default. Catalogs and event taxonomies differ from one tracking system to the next, and the behavioral foundation models in Table 1 are built accordingly: one per platform, with a head fitted per task inside it. If that heterogeneity is fundamental, a model spread across retail, engagement, and payments should be mediocre at each. In our results it is not, which weakens the case for one model per domain. We expect Large Behavioral Models to take the place in behavior-driven industries that LLMs now hold in language. The reason is accuracy as well as convenience: on our benchmark, one model outperforms both the specialists it would replace and the general-purpose alternatives, prompted LLMs and Large Tabular Models. We therefore treat behavior as a modality in its own right, not as a table or a text corpus.

Our main contributions are as follows:

1. **Zero-shot transfer that beats fully trained specialists.** Under standardized full-catalog ranking with temporal splits on two public datasets held out of pretraining, short video (PixelRec) and reviews (Amazon Sports), BehaviorGPT-v4 *without any samples* from the target outranks SASRec, HSTU, and RecFormer trained on the target’s full training split, OneRec 8B Pro, a prompted LLM (Gemma 4 31B), gradient-boosted trees (LightGBM), and a Large Tabular Model (TabPFN v3). No baseline reaches our zero-shot accuracy at any budget (Figure 1). Zero-shot behavioral transfer has been shown before (Section 6.4), but not at industrial pretraining scale, across industries, or against specialists trained on the full target dataset rather than a fraction of it. BehaviorGPT-v4 was never trained on reviews or videos at all.
2. **Breadth from one pretrained model.** With no per-dataset architecture, objective, or vocabulary changes, one fixed BehaviorGPT-v4 model leads the strongest specialist on 30 of 31 dataset–task pairs with no target

Model	Pretraining corpus	Par.	Data	Ind.	Tasks	0-shot	Carrier	Geom.
<i>Industrial behavioral foundation models</i>								
HSTU [112]	Meta: engagement	0.13B ^e	100B ex.	1	Multi*	No [†]	IDs	Behav.
Netflix FM [82, 107]	Netflix: video	1B ⁿ	2T tok. ⁿ	1	Multi*	No ^c	IDs + meta.	Behav.
Shopify CFM [3]	Shopify: storefronts, Shop app	<8B ^s	~1T tok. ^s	1	Multi*	No	IDs	Behav.
PRAGMA [83]	Revolut: banking	0.91B ^e	24B ev.	1	Multi*	No	Schema + text	Behav.
nuFormer [10]	Nubank: card txns.	0.30B ^e	500B tok.	1	Multi*	No ^m	Text + schema	Behav.
TREASURE [109]	Visa: card txns.	~0.005B ^e	6B txn.	1	Multi*	No	IDs + schema	Behav.
TransactionGPT [29]	Visa: card txns.	0.01B	1B+ txn.	1	Multi*	No	IDs + schema	Behav.
Stripe PFM [65]	Stripe: payments	—	10s B txn.	1	Multi*	No	Schema ^u	Behav.
TradeFM [63]	J.P.M.: trade flow	0.49B ^e	10.7B tok.	1	Multi*	Yes ^g	Schema ^g	Behav.
<i>Transferable recommenders</i>								
ZESRec [27]	Public rec. corpora	0.001B ^f	2.3M int.	1	1	Yes	Item text	Lang.
UniSRec [50]	Public rec. corpora	0.003B ^f	1.4M seq.	1	1	Yes ^z	Item text	Lang.
VQ-Rec [51]	Public rec. corpora	0.003B ^f	1.4M seq.	1	1	Yes ^z	Item text	Lang.
RecFormer [72]	Public rec. corpora	0.11B ^e	3.6M seq.	1	1	Yes	Item text	Lang.
RecGPT [60]	Public rec. corpora	0.021B ^f	12.5M seq.	1	1	Yes ^z	Item text	Lang.
<i>LLM-backed generative recommenders</i>								
OneRec-Fdn. [115]	Kuaishou: video, ads, product	6.95B ^e	130B tok.	1	Multi	No [#]	Item text	Lang.
Prompted LLMs [53, 75]	(LLM pretraining)	—	—	Multi	Multi	Yes	Item text	Lang.
<i>Large Tabular Models</i>								
TabPFN [43]	Synthetic causal graphs	0.026B ^h	100M tab.	Multi	1 [¶]	No ⁱ	Synth. prior	Synth.
XTab, CARTE [117, 68]	Public tables; YAGO3	0.009B ^h	18M trip.	Multi	1 [¶]	No	Schema text	Lang.
TabuLa-8B [38, 45]	Serialized tables	6.98B ^h	4M tab.	Multi	1 [¶]	Yes	Row text	Lang.
BehaviorGPT-v4	Retail, engagement, pay.	12.5B	150B act., 3.6T tok.	Multi	Multi	Yes^g	Multimodal	Behav.

Table 1: Where behavioral foundation models stand. No specialized behavioral model is broad in both industries and tasks. The general-purpose alternatives, Large Tabular Models and prompted LLMs, are not competitive in accuracy (Figure 1) or latency (Table 3). *Par.*: **non-embedding** parameters, the only count comparable across architectures, since ID-based models hold most of theirs in per-item lookup tables that grow with the catalog; dashes mean not reported, so our figures are the largest *disclosed*. *Data*: pretraining corpus in the unit each work reports. *Ind.*: industries in pretraining (*Multi*: not tied to one company or vertical). *Tasks*: task families one pretrained model is evaluated on. *0-shot*: evaluated on a dataset with no samples from it. *Carrier*: what an event enters the model as, and so what can carry it to an unseen catalog. *Geom.*: what makes two events close in the learned space. Appendix A.1 defines the columns and marks; Section 6.4 develops the comparison.

data (Figure 11), and fine-tuned it leads all 31, over 12 catalogs and a cross-merchant payment stream (Figure 5). We are aware of no behavioral model previously shown to hold up across industries, subdomains, and task families at once; prior work is broad along at most one of those axes (Table 1). Two exceptions are broad by construction: a prompted LLM, whose breadth is inherited from language rather than learned from behavior, and a Large Tabular Model, which applies to any domain but is not built for behavior: it cannot rank even moderately sized catalogs or represent user sequences well. We outperform both on every task in our benchmark that can be posed to them.

- Generality that grows with model scale.** From our 0.5B model to our 12.5B, a $25\times$ increase in parameters, zero-shot transfer improves by a geometric mean of $1.94\times$ across the same grid of datasets and tasks, and the larger model keeps a $1.47\times$ margin once both are given the target’s own data (Figure 12).
- Real-world deployment evidence.** Multi-year A/B tests of our BehaviorGPT models against incumbent commercial systems in search, recommendations, and physical retail show that the approach holds under production latency and cost constraints, which bind hard in these industries: a ranking call sits inside a page load, and a low-margin conversion must pay for the inference behind it. The two established routes to generality break these constraints: prompted LLMs and semantic-ID generative recommenders both decode in multiple steps, and prompted LLMs rank only a shortlist supplied in context. BehaviorGPT-v4 keeps the single-step retrieval cost of a classical recommender while transferring across catalogs (Table 3), and on our benchmark it is the most accurate method at that cost (Figure 3).
- Method and design decisions.** An account of what training a foundation model on discrete action spaces at this scale involves: multimodal event representations, two-stage embedder–contextualizer training, loss and candidate-sampling design, and exact softmax over catalogs of 10^9 candidates, together with the internal experiments and ablations behind each choice (Appendices B and G).

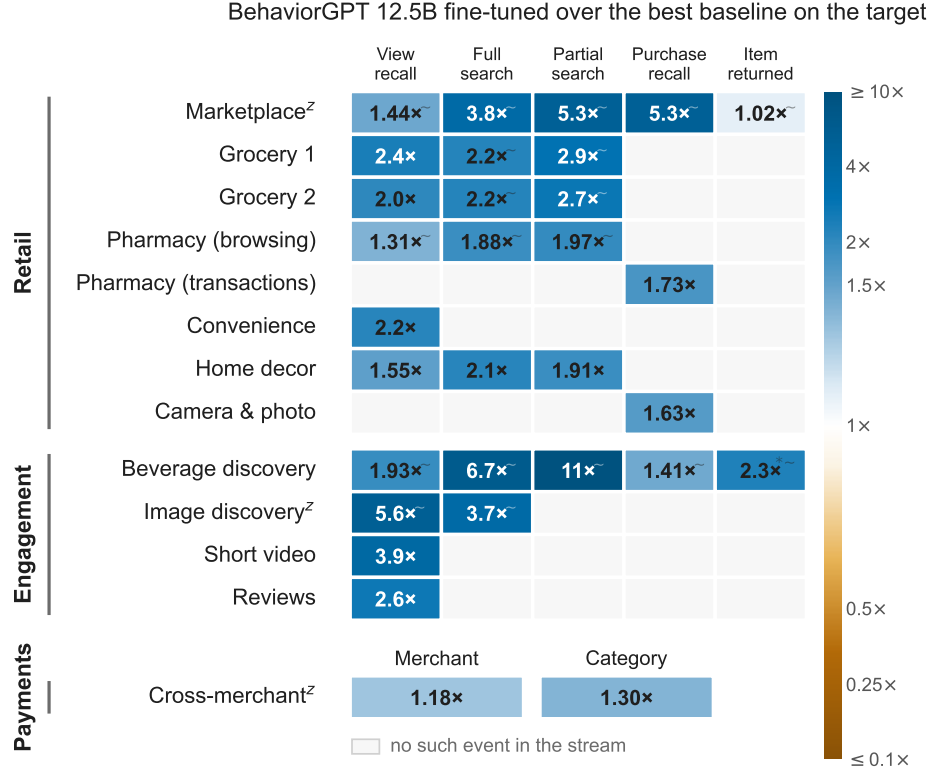


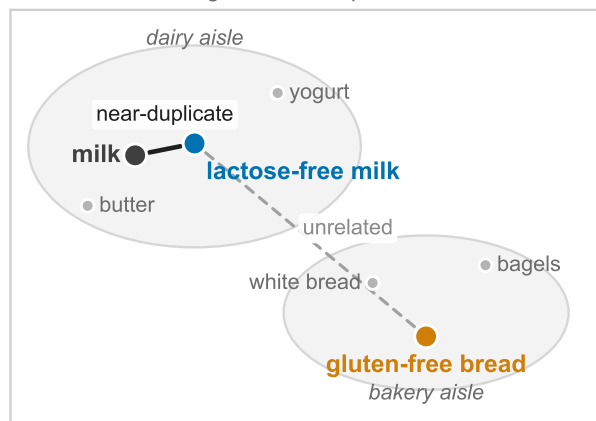
Figure 5: **Adapted where evaluated, one pretrained model beats every baseline on every dataset and task.** Each cell is the 12.5B’s recall@10 over the strongest baseline’s on the same positions, fine-tuned on the target’s training split on 22 cells and unadapted on those marked ^z (marketplace, image discovery and payments, which have no fine-tune). The baseline is the highest of every baseline run on that cell rather than one fixed opponent (Table 9); Appendix F.1 explains the marks. We lead all 31 populated cells, by a geometric mean of 2.4×, and all 22 fine-tuned cells by 2.35×; with no adaptation anywhere, one fixed model leads 30 (Figure 11).

- Availability.** An SDK and API that deploy a new catalog zero-shot, an example notebook that reproduces a zero-shot PixelRec evaluation through the API, and live demos built on the SDK, at www.unboxai.com/behaviorgpt (Section 4; demos in Appendices I and J.1).

Roadmap. Section 2 describes the approach (the framing of behavior as sequences of multimodal events, the training recipe, the data, and the evaluation protocol), with full method details in Appendices B–D. These appendices are a deliberate record of the recipe: they collect design lessons from six years of building these models, including internal runs reported in less detail than the main results. The benchmark appendices (Appendix E onward) document the evaluation protocol and results in full. Section 3 presents public-dataset, breadth, scaling, and deployment results, with the internal experiments and ablations behind the recipe in Appendix G. Section 6 reviews related work and locates our claim relative to both the industrial and the academic transfer literature. Two companion papers cover the position and its implications [33] and compute-optimal scaling for this model family [37].

LLMs / vision: features

Similar names, images and shelf placement are close

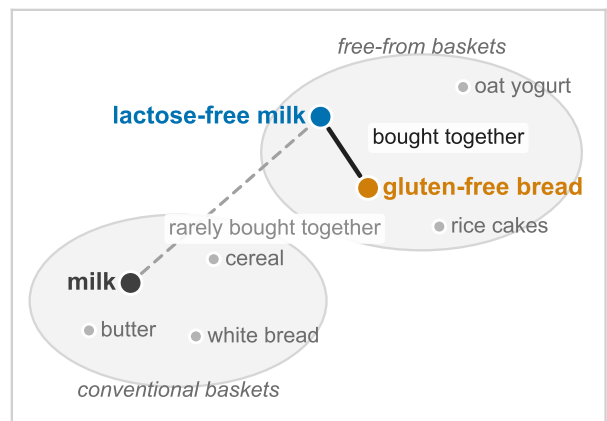


What the model sees: a description of lactose-free milk

"milk with its sugar, lactose, removed or broken down"

Behavior: actions

Items acted on in the same sessions are close



What the model sees: a session

lactose-free milk → gluten-free bread → olive oil

Figure 6: Description and behavior disagree on similarity. Regular and lactose-free milk are near-duplicates in names, images, and shelf placement, but nearly disjoint in behavior: few users purchase both, and lactose-free milk co-occurs with gluten-free bread far more than with regular milk.

2 Approach

This section summarizes the recipe needed to interpret the results: what we require of the model (Section 2.1); behavior as chronological sequences of multimodal events, and the two-module model that consumes them (Sections 2.2–2.4); the losses and data it is trained on (Sections 2.5–2.6); adaptation and serving (Sections 2.7–2.8); and the evaluation protocol (Section 2.9). Appendix B records the full set of techniques and design decisions.

2.1 Desiderata

We first state what we require of the model, since the next-event objective alone does not determine several of the design choices that follow.

Predictive requirements. We ask pretraining for a single model that predicts *any* action from a history of actions:

$$p(\text{action} \mid \text{history of actions}, \text{action} \in \text{candidate set}). \quad (1)$$

A history is a chronological sequence of past events together with the contextual attributes (market, age band, store format) under which they occurred; these are part of the history even where they occupy no sequence positions (Section 2.2). The candidate set fixes the support: (1) is the model’s distribution over all actions, restricted to that set and renormalized. Unlike a language model’s vocabulary, the support is not fixed in advance. Candidates enter through their multimodal features rather than an identifier, so a candidate set may contain actions absent from pretraining, which is what makes an unseen catalog rankable at all. It may hold a full catalog of up to $\sim 10^9$ items or a handful, and normalizing over the largest is a central computational problem of training (Section 2.5). Nor must it come from the domain that generated the history: given a partial grocery basket and a candidate set of vehicles, the model must return a distribution over vehicles. A task is thus specified by a history and a candidate set rather than by a choice of model, and the same weights serve every such query; in deployment, “customers who bought this also bought” and “customers who viewed this also viewed” differ only in whether the prompt ends in a purchase or a view.

We also want the reverse direction: which history most plausibly precedes a given outcome, such as the search most likely to have preceded a purchase. This is a capability rather than a pretraining objective: pretraining is causal, and masked fine-tuning supplies the reverse direction afterwards (Section 2.5).

Representational requirements. Most tasks use the model end-to-end, but its intermediate representations also matter: they make predictions explainable and serve uses not posed as (1) at all (Appendix J). Pretraining should therefore yield, as a byproduct, four metric spaces in which distance reflects behavioral co-occurrence rather than descriptive similarity (Section 1):

- (D1) over actions and the items acted upon, so that products users treat interchangeably are close;
- (D2) over agents, so that users with similar behavior are close;
- (D3) over domains, so that markets, age bands, or store formats with similar behavioral structure are close;
- (D4) over actions *conditioned on* an agent, since interchangeability depends on the agent: which of the two milks in Figure 6 substitutes for the other is a property of the user, not of the pair.

These should emerge from the pretraining objective itself, not from separately trained heads. D3 follows from the domain conditioning of Section 2.2; Appendix J describes how agent representations are read out of the trained network.

2.2 Behavior as Sequences of Events

Ordering an individual’s actions chronologically yields a sequence structurally analogous to a sentence, which admits the same causal, autoregressive treatment. Where graph-based recommenders model relations among products or users explicitly, we capture them through the temporal structure of what each user did next, without materializing a graph.

The atomic unit of the sequence is a single action, represented by its text, imagery, and numerical fields rather than by an identifier. Where this boundary is drawn decides what gives an action its meaning: the other tokens in its own description, or the other actions in the same user’s history. Section 2.3 reports the experiment that separates the two.

To the sequence we add *domain* information: contexts spanning multiple events, such as market, store format, or demographic attributes (Figure 7), encoded following CTRL [66], which conditions generation without consuming sequence positions. Domains raise CartMetric accuracy (Appendix D.6) from 0.35 to 0.386, close to 10%, with most of the gain on users with fewer than 50 recorded events, where behavioral history is thinnest (Appendix G.2). Conditioning this way also places the domains themselves in a metric space, D3 of Section 2.1.

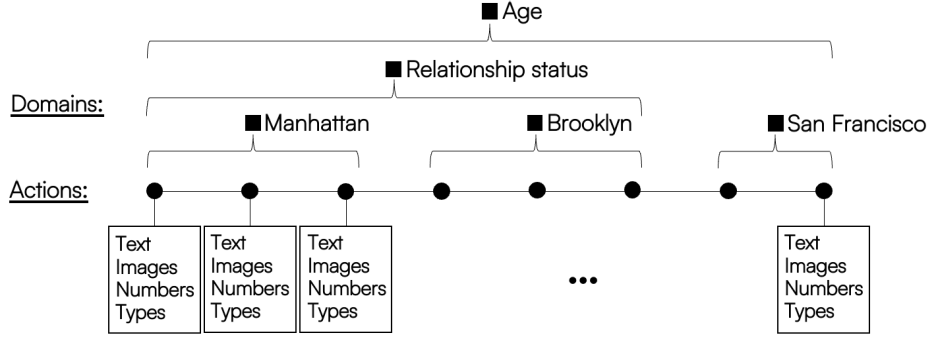


Figure 7: Behavioral sequences with domains. Events are ordered chronologically and each carries multimodal features. Domains are contexts spanning one or more events (market, store format, demographic attributes) and enter as conditioning rather than as additional sequence positions, so they cost no context length.

2.3 Events as Multimodal Features

Event IDs as tokens. The naive approach treats each distinct event as a token and predicts the next event ID, as in language modeling. It fails for three reasons, which is why events enter our model through their features instead (the *carrier* of Table 1). First, IDs do not survive a tracking boundary: systems use different conventions for the same product, so an unseen catalog maps to the wrong embeddings or to none; unsupervised translation between ID spaces [70] can align them, but only where both are observed. Second, IDs pool no statistical strength: most events are rare, and a model that sees each as an arbitrary index cannot share what it learns across them, whereas descriptions let a preference for a price point or a color carry over to every event that shares it, and from one dataset to another. That transfer, rather than warm- or cold-start performance, is the main advantage of features over IDs, and what lets one model make use of diverse corpora. Third, IDs offer no feature-level handle for explaining a prediction or for generating product text and imagery.

Embeddings. Training the text and image embedders end-to-end on behavior works better than adopting pre-trained ones (Appendix G.3), and since behavioral text (short search terms, brand names, merchant descriptions) is unlike internet text, we train the tokenizer on it as well. Nor should the embedders be trained apart from the sequential task: auto-encoding objectives fitted to individual events, detached from their place in a user’s history, yield weaker representations.

The granularity of masking shows why. Consider a user’s basket, rendered as descriptions:

pasta 1kg, barilla, \$7.3 <sep> rao’s tomato basil sauce, \$6.8 <sep>

Masking individual tokens and masking whole descriptions both instantiate the distributional hypothesis (“you shall know a word by the company it keeps” [31]), but they disagree about what the company is. Token-level masking is largely solvable from the rest of the same description, so it teaches naming conventions; masking the whole description leaves only the other products the user bought, so it teaches the cross-product associations that behavior carries. We find the second yields better embeddings and recommendations, so the event, not the token, is our unit of prediction. The pretraining objective itself remains causal (Section 2.5).

Features of events and actions. Behavioral frequencies are long-tailed, and the features attached to events (images, descriptions, prices) are typically static and human-authored, which invites memorization over generalization. Overfitting takes two forms: to features that identify a tracking source rather than a behavior, and to the particular feature values seen in training, which leaves the model unable to handle new or altered events. The model therefore needs combinatorial generalization [81] over unseen events and unseen feature combinations, a stronger demand than a language model faces, since its vocabulary is fixed in advance. Data augmentation [32] is central to meeting it (Appendix G.7).

Two-stage training. The model has two modules: the *embedder*, which maps an event’s text, imagery, and numerical fields to a vector, and the *contextualizer*, which consumes the sequence of event embeddings and produces the distribution of (1) (Figure 8). Training the embedder end-to-end is expensive, since it runs on every candidate as well as every event, so we train in two stages. Stage 1 trains both modules against the sequential objective, so that event representations are shaped by behavior rather than by an auto-encoding proxy. Stage 2 freezes the embedder and continues training the contextualizer alone; freezing lets candidate embeddings be precomputed and cached instead of

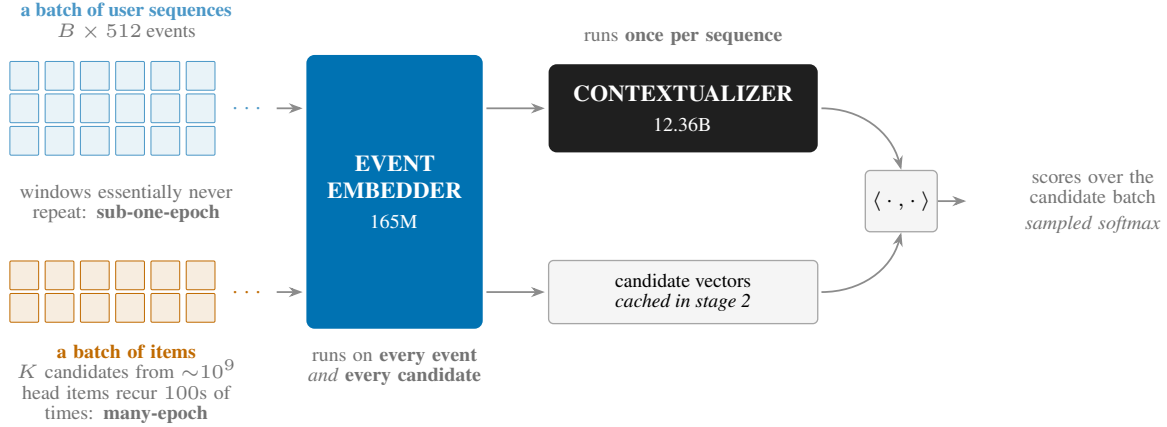


Figure 8: **The two modules, and the two batches that train them.** Every event, in the sequence and among the candidates, enters as its multimodal features through the *embedder*; only the sequence batch reaches the *contextualizer*, which runs once per sequence and scores candidates by a single product against their vectors. Stage 1 trains both modules jointly with in-batch negatives; stage 2 freezes the embedder, caches the candidate vectors, and trains the contextualizer against pools drawn from the full catalog (Section 2.5). The annotations show the asymmetry behind the compute accounting: an embedder parameter is billed on every event and every candidate, a contextualizer parameter once per sequence, and sequence windows essentially never repeat while head items recur hundreds of times [37].

Module	Layers	Width	Parameters
Embedder (stage 1, then frozen)	8	768	165M
Contextualizer (stages 1 and 2)	64	4096	12.36B
Total			12.5B

Table 2: Configuration of the BehaviorGPT-v4 checkpoint. The embedder is frozen after stage 1, so candidate vectors can be precomputed and cached for both training and serving (Sections 2.3 and 2.8).

recomputed every batch, which is what makes stage 2’s large candidate pools affordable (Section 2.5). The split of parameters between the two modules is one of the axes of our scaling study [37].

Language as a special case. The embedder–contextualizer structure contains the language model as one instance: let every event be a single token, the embedder a lookup table, and the candidate set a fixed vocabulary, and the contextualizer becomes a decoder-only transformer trained by next-token prediction. The same model does train on text alone, with segments of text as events (Appendix C.4). An LBM lifts both restrictions: events are described by open-ended text, imagery, and numbers, and the model predicts a distribution over any catalog, including one never seen in training. The reduction also runs the other way. Because candidates are embedded from their features, a candidate set can hold anything the embedder can describe (a complete answer, a query, a multimodal action), so any task whose answers can be enumerated becomes a single step: one contextualizer pass and one product against cached candidate vectors (Section 2.8), where an LLM would decode its answer token by token.

Predicting actions costs far more than reading them. A language model scores a vocabulary of hundreds of thousands of tokens by lookup; the action space holds tens of billions of events, each embedded from its own text and imagery, so scoring C candidates costs up to C times as much as encoding the input sequence. This asymmetry drives several further design choices, detailed in Appendix B: alternative target objectives, temporal and popularity features injected at the logit level, candidate caching, and learned quantization.

2.4 Model, Scale, and Training Configuration

Table 2 gives the configuration of the two modules. The embedder is 8 layers at width 768 and the contextualizer 64 layers at width 4096, and the contextualizer holds 99% of the parameters: describing an event is a far smaller problem than modeling a sequence of them.

Scaling laws for LBMs are the subject of a companion paper [37], summarized in Section 3.3. One difference from LLMs bears directly on configuration: an LLM’s vocabulary is fixed and varies little across models, whereas the candidate space is a hyperparameter in its own right, with bottlenecks of its own. Its effect on difficulty cuts both ways: more candidates make each next-event prediction harder, but finer-grained candidates also make each history more distinctive, so earlier events say more about later ones [37].

2.5 Losses and Candidate Sampling

An LLM computes a full softmax over a fixed vocabulary at every step. An LBM cannot, because its candidates are not fixed in advance (Section 2.1) and three scales are at play: the action space the model is trained over, tens of billions of actions across all corpora; a single catalog, up to $\sim 10^9$ items; and the candidates any one loss computation involves. These are usually sampled, and the sampling distribution then *defines* the loss and what the model learns; where a full catalog must be scored, an exact softmax handles up to 10^9 items. Five design decisions follow, developed in full in Appendix B.5.

Local and global losses. Batch-derived candidates over-represent hard negatives and are drawn in proportion to popularity, so their loss divides popularity out of what the model learns. We therefore pair a *local* loss over batch candidates, which teaches fine discrimination between similar items, with a *global* loss over candidates sampled from the full space, which teaches the overall action distribution. Stage 1 uses the local loss alone, over the batch candidates of all devices gathered together; stage 2 adds the global loss, drawing candidates from the catalog cache that the frozen embedder makes possible. More candidates help monotonically, and the compute-optimal number is swept in our scaling study [37].

Separate softmaxes per distribution. A single softmax over mixed candidate distributions (data sources, markets, or time periods) teaches the model to separate them, the opposite of the alignment that cross-domain transfer requires. We keep one softmax per distribution and weight the resulting losses, which aligns the embedding spaces instead; at 24,000 candidates this is worth more than doubling the candidate count without it (Table 48), and it is what lets out-of-industry data help rather than hurt (Section 3.4). The rule applies to genuinely distinct distributions: candidates from one distribution, merely sharded for computation, belong in one softmax, and fully disjoint distributions forfeit the transitivity that shared items provide.

Exact softmax at catalog scale. Where a full catalog must be scored, we compute an exact softmax cross-entropy over up to 10^9 items, roughly ten thousand times an LLM vocabulary. An optimized streaming kernel for our distributed setting applies online softmax [80], in the spirit of fused large-vocabulary cross-entropy kernels [102]: a log-sum-exp over vocabulary shards with a hand-derived backward pass that never materializes the full logit matrix (Appendix D.4). The action space as a whole remains beyond an exact softmax, so the sampling design above still governs pretraining.

Simple objectives at scale. We stay close to step-wise autoregression, predicting one or two targets per step. Richer multi-target objectives can reach a better loss, but we find the simpler ones win at equal compute at scale, the pattern language modeling settled into.

Causal pretraining, masked fine-tuning. Pretraining is causal. Masked objectives capture useful bidirectional conditioning, such as which search most likely preceded a purchase, but converge more slowly at scale, so we apply them as fine-tuning after pretraining.

2.6 Data

BehaviorGPT-v4 is pretrained on 150 billion actions, or 3.6 trillion tokens, of retail, engagement (app usage), and payments behavior, in decreasing order of share. An action averages roughly 24 tokens because it is serialized as its multimodal features (description text, categorical fields, numerical values such as price) rather than as an identifier (Section 2.3), so action and token counts measure different things, and we report both.

The corpus combines three families of events: transactions (purchases, returns, individual receipt lines); interactions (clicks, likes, searches, and other engagement with items); and contextual attributes of a user or session (location, age band, market), which enter as domains (Section 2.2). A subset of sequences also carries demographic attributes and outcome labels such as fraud indicators.

Behavioral corpora are far less homogeneous than text corpora. A language model fitted to diverse internet text learns $p(\text{language})$ and, implicitly, conditionals such as $p(\text{language} \mid \text{author})$ or $p(\text{language} \mid \text{country})$. Behavior adds a conditioning variable with no counterpart in text: what gets recorded depends on the system that recorded it, so a corpus supplies $p(\text{behavior} \mid \text{tracking system})$, each system a dialect with its own event taxonomy, catalog, and convention for

what counts as an interaction. The per-distribution softmaxes of Section 2.5 exist to align these dialects rather than let the model separate them.

Behavior is also bounded by what an interface permits. A poor search interface yields sequences in which users never found what they wanted, and a model fitted to them learns the constrained behavior rather than the intent behind it. Part of the problem is measurement: in one deployment, adding an event recording whether a user left the search page before clicking changed what the model inferred about search intent and made its search predictions markedly less diffuse.

Three properties matter more than raw size in making a corpus learnable: events that carry predictive signal about one another, enough distinct event types co-occurring for the model to learn relations across types, and sequences long enough in elapsed time to expose repeat and seasonal behavior. Augmentations, auxiliary text and internet data, and time-delay encoding are described in Appendices G.7, C.4, and B.4.

The recipe also extends beyond consumer data. Workforce event streams differ from this mixture more than its three industries differ from one another, with a far smaller event vocabulary and a coarser time granularity, yet a separate 5M-parameter model trained from scratch on 43M workforce events with the same architecture and objective predicts employee attrition at 91% accuracy (F1 0.93) [34] (Appendix C.3). That is transfer of the method, not of this checkpoint, and it makes workforce behavior a candidate for a future mixture (Section 5).

2.7 Fine-tuning

Pretraining runs once; adapting to a new dataset or deployment is a much smaller step. We fine-tune a small subset of the parameters, either LoRA adapters [55] or the last transformer block with the final normalization, output projection, and domain embedder, depending on the target, running stage 1 and then stage 2 on the target’s data with early stopping on a small validation set. An unseen catalog needs none of this: its items are embedded from their features and enter the candidate set without training (Section 2.3), so adaptation changes how well the model ranks, not what it can rank.

We fine-tune only when a target is absent from pretraining or its own data is plentiful enough to warrant it; deployments already represented in the pretraining corpus are served the checkpoint as-is. Because the delta is small, one base model serves every deployment, with per-deployment deltas swapped in where they exist, and many adapters can be served concurrently at low marginal cost [19].

Fine-tuning reuses the pretraining formats, so inference formats still match training. EMA-anchored adaptation and distillation from the stage-1 model were tried and not adopted (Appendix B.6). Post-training for reasoning and test-time compute in the behavior space, by supervised fine-tuning and Group Relative Policy Optimization (GRPO) [25], is described in Appendix B.8.

2.8 Inference

Serving cost is a design constraint in these industries. A ranking call sits inside a page load, and where margins are thin, the inference behind a decision is a material fraction of what that decision is worth. Table 3 compares the routes to a ranked list by the number of sequential passes through the model per request, which does not depend on hardware, alongside measured latency.

A classical sequential recommender is as cheap as ranking gets: it encodes the history once and scores candidates with one matrix product, but its embedding table cannot leave the platform it was built on (Table 1). The two established routes to *generality* both give up that single step, and ours does not. Semantic-ID generative recommenders and prompted LLMs reach new items by decoding in multiple steps, and prompted LLMs rank only a shortlist supplied in context; for actions, the beam search of semantic-ID decoding does not even buy accuracy, as greedy decoding scores higher (Table 50). Because our embedder is frozen (Section 2.3), candidate vectors are precomputed and cached, so a request costs one contextualizer pass and one product against the cache, whatever the candidate’s modality, and a new item enters the cache by one embedder pass over its features, with no retraining. Figure 3 sets these costs against accuracy.

2.9 Benchmarks and Evaluation Axes

Most public benchmarks test one narrow form of generalization: the next item for a known user in a known catalog, at a time inside the data window. A foundation model for behavior must generalize along six axes, which structure both our internal evaluations and the public-dataset protocol of Section 3.1:

1. **New time**: generalize to time periods after the training window (temporal splits are mandatory, not optional).
2. **New items**: generalize to products and actions never seen in training, via their multimodal features.

Route to a ranked list	Passes	Full catalog	ms/query
Count-based rules (popularity, recency)	0	Counts and recency only	0.04 (0.02–1.4)
ID-based sequential (SASRec, HSTU)	1	Own catalog only	0.2 (0.02–1.6)
Text-encoder retrieval (RecFormer)	1	Yes	5.7 (0.94–19)
Per-candidate LTM (TabPFN)	n	No, shortlist only	3.7 (0.94–57)
Per-candidate trees (LightGBM)	n	No, shortlist only	0.59 (0.41–0.84)
Semantic-ID generative (TIGER, OneRec)	mb	Approximately	20 (0.36–38)
Prompted LLM (ranking in context)	T	No, shortlist only	34 (3.6–186)
BehaviorGPT-v4	1	Yes	0.71 (0.06–2.4)

Table 3: What a ranked list costs to produce. *Passes*: sequential forward passes through the model per request, the part of serving cost set by the architecture rather than the hardware; n is the shortlist size, m and b the code depth and beam width of semantic-ID decoding ($mb=40$ at typical settings), and T the number of decoded tokens. *Full catalog*: whether the list is drawn from the whole catalog, as Figure 1 requires. *ms/query*: median and range of harness throughput over the benchmark’s runs, not serving latency, so the column is read down rather than across rows. Appendix F.3 gives per-route details and measurement caveats.

3. **New users**: generalize to individuals not seen in training.
4. **New domains and markets**: generalize to entire catalogs, merchants, or countries never seen in training (the zero-shot regime).
5. **Dropped or short history**: perform well under cold-start, without assuming long histories.
6. **New tasks**: generalize across task families (from the next purchase to the next reviewed item, the answer to a search, or an attribute of the next event) without a head fitted per task.

One rule cuts across all six axes: nothing an arm reads (history, features, or any statistic) may come from after the timestamp of the position being scored. Anything from before it is fair, including behavior aggregated *across* users up to that moment (popularity, trends, the drift of a catalog’s traffic), which is exactly what a deployed system has. Pretraining obeys the same rule: it is causal in time, and the time-since and popularity trajectories of Appendix B.1 are such aggregates, computed up to each position.

Figure 1 exercises axes 1–4 and 6 on two datasets held out of pretraining, and the breadth results of Section 3.2 mix axes 2–4 across twelve catalogs in three industries. Axis 5 is present throughout, since evaluations include histories of every length, but is not a focus.

Comparisons to LLMs. The LLM arm is Gemma 4 31B [40], served locally as a zero-shot and N -shot next-item ranker: for each target position it receives the user’s 20 most recent interactions and a numbered list of the same candidates every other bounded arm ranks (Appendix F.4), all rendered as catalog text, and returns a ranked JSON array under greedy decoding. The N -shot variant adds N (history, next-item) pairs from the training split to the prompt, with no parameter updates; Appendix F.5 gives the full protocol.

3 Results

3.1 Public Datasets

This subsection tests the first headline property, *zero-shot transfer*: with *no* samples from a public sequential-recommendation dataset, BehaviorGPT-v4 outperforms per-dataset models trained on it, and it improves further with light fine-tuning. We test on public data because only there can the claim be independently verified, and because prior industrial behavioral foundation models have not reported this setting (Section 6.4). An example notebook runs our zero-shot PixelRec evaluation through the public API (Section 4).

Protocol. Figure 1 reports two datasets: PixelRec [21] (Pixel200K subset), short-video recommendation over cover images, and Amazon Reviews: Sports and Outdoors [52] (Amazon Sports), retail. Each method’s score is divided by our zero-shot score on that dataset before the two are averaged, so the average does not mix dataset difficulty; Figure 9 shows each dataset on absolute recall. Ranking is over the full catalog with no sampled negatives, on a temporal split rather than leave-one-out [69, 59], with unseen items and users, and every arm scores identical positions. The metric is Effective Recall@10: a target an arm structurally cannot reach (outside its candidate pool, or absent from its training interactions) counts as a miss, as it would in deployment. We compare our checkpoint with no target data, the same checkpoint fine-tuned on the target’s training split, and baselines run as their authors intend, trained on that split where they train at all: SASRec and HSTU, RecFormer, OneRec 8B Pro, LightGBM, TabPFN, a prompted Gemma, and a popularity rule. Section 3.2 widens this to 12 catalogs under a second protocol.

Results. We draw three findings. First, zero-shot transfer beats trained specialists: with no target data we lead every arm that runs without it, and no baseline reaches our zero-shot score at any budget; given all 2.9×10^6 and 2.5×10^6 target samples, the best stops at $0.62\times$. Second, pretraining changes what a target sample is worth. RecFormer, built around item text, has the best baseline zero-shot score ($0.47\times$ ours) but stops improving by 36,000 samples, plausibly because its text carries little of what the target’s behavior can teach (Appendix E); our checkpoints improve at every rung, the 12.5B to $2.18\times$ and the 0.5B to $1.67\times$ the 12.5B’s zero-shot score, averaged over the two datasets. A strong starting point and the ability to convert target data are separate properties, and a foundation model needs both. Third, scale helps: the 12.5B leads the 0.5B zero-shot and at every budget, and the 0.5B needs 9,949 to 35,928 target samples to reach what the 12.5B scores with none. Appendix E gives arm-by-arm numbers and adaptation protocols.

What each dataset asks of transfer. Both datasets are held out of pretraining, but at different distances. Amazon Sports is the nearer transfer: the review behavior is new, but its items are retail products, the largest share of our mixture. PixelRec is the farther one: we hold no short-video data, so the transfer crosses industries. It is also a *pixel* dataset, built to test whether raw visual content can replace item IDs, and every transferable recommender (Section 6.4) represents items by text, so it tests a carrier those baselines lack.

Limits of the comparison. Pretrained models differ in their pretraining data, so the figure compares systems as fielded, not architectures under a control. One such confound favors a baseline: RecFormer is pretrained on other Amazon Reviews categories, while BehaviorGPT has seen no Amazon Reviews data, yet BehaviorGPT outperforms it there. Parameter count alone does not explain the ordering either: OneRec, at $73\times$ RecFormer’s size, scores $0.10\times$ zero-shot against RecFormer’s $0.47\times$, and within our family $25\times$ the parameters buys a factor of 1.3 after adaptation. Figure 12 extends the scale comparison to every dataset and task.

3.2 Breadth: One Pretrained Model Across Industries and Tasks

The second headline property is breadth from one pretrained model. Figure 2 shows it most directly: every method on every dataset it ran on, fifteen arms over twelve catalogs on the one slice all methods share, as a leaderboard rather than a ratio against us, so each score is divided by the dataset’s best rather than by a baseline we pick. Our four checkpoints lead all twelve datasets and both payments tasks; the eleven baselines lead none. No baseline reaches two thirds of the leader on any catalog, the best being HSTU at 64% on pharmacy (transactions). The only baseline cells above four fifths are in payments, where two session rules reach 85% and 83% of our next-merchant score, because merchants repeat within a session far more than products do. The zero-shot 12.5B, given nothing from the target, sits at a median 83.1% of the leader and leads two datasets outright, above every baseline trained on the target, extending the result of Figure 1 from two datasets to twelve. Behind us, the four strongest baselines are within a rank of each other (RecFormer 7.3, HSTU 7.9, Gemma 4 31B and SASRec 8.1), so a prompted LLM restricted to a thousand candidates ranks with three models trained on the target. Split by task (Figure 10), four different arms are the strongest baseline across the seven columns: Gemma on the two search columns, by reading the query; a session rule on item returned and payments; the

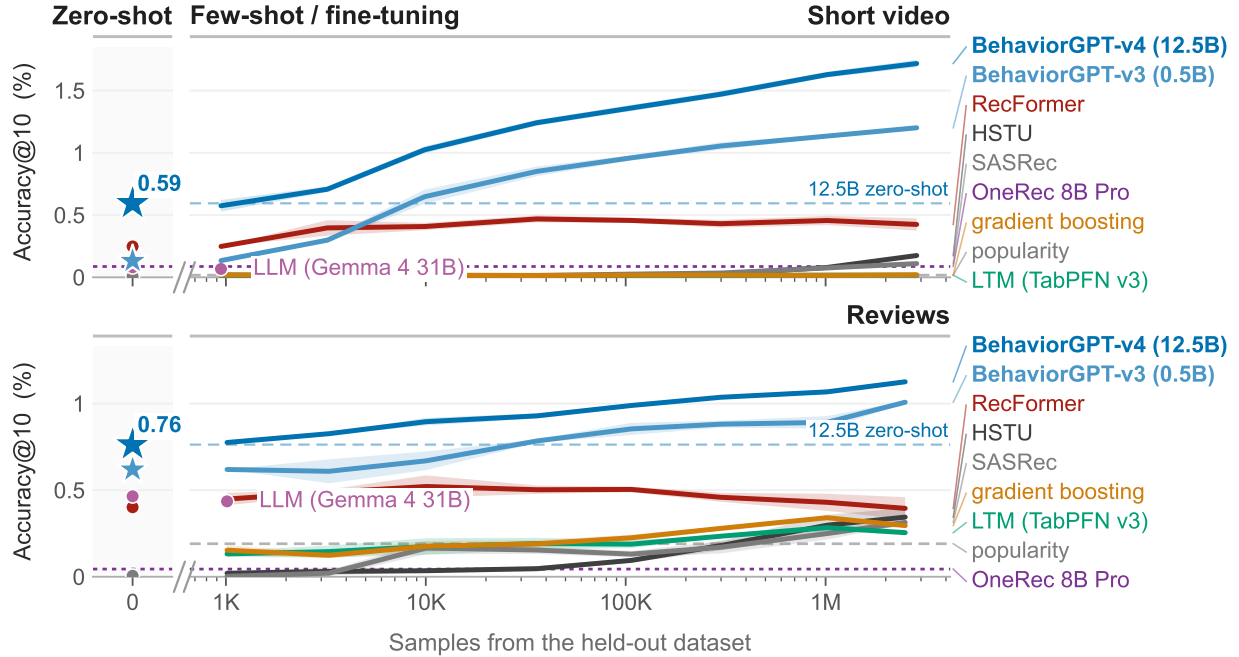


Figure 9: **Figure 1 drawn one dataset per row rather than averaged, on absolute recall.** Short video above (PixelRec; engagement, 89,444 items, 79,399 held-out targets) and reviews below (Amazon Sports; retail, 1,587,428 items, 69,723 targets), with separate vertical axes. Curves are 5-seed means with ± 1 s.d. bands; zero-shot inference is deterministic. Popularity is counted on the target’s training stream, so it appears on the fine-tuning axis only; OneRec needs no target data and spans both panels. Three things show only per dataset: the text-reading baselines’ zero-shot scores track how descriptive item text is; on short video, adaptation first costs the 12.5B accuracy, recovered by 3,219 samples; and the 0.5B is within $1.2\times$ of the 12.5B on reviews, near our pretraining mixture, but $4.4\times$ behind on short video. Appendix E gives arm-by-arm numbers.

trained specialists on views and purchases. No baseline family is strong everywhere. Appendix F.8 reads both figures arm by arm.

A leaderboard hides the size of the margin and pools every task into one slice. Figure 5 shows both, across 12 catalogs and five tasks, with no per-dataset changes to architecture, objective, or vocabulary. Each cell faces a different opponent, the strongest we ran there: on 12 cells a model trained on the dataset’s full training split, on nine a 31B-parameter LLM, and on ten a rule reading the session in front of the target. We lead every cell, by up to $11\times$; the narrowest margin, $1.02\times$ on marketplace returns, reflects a ceiling: a returned item is by construction one the session already holds, so a simple rule can nearly match us. Without target data the same weights still lead all but one cell (Figure 11), on users the model has never seen and on catalogs wholly or partly absent from pretraining, which a per-dataset specialist cannot do by construction.

The baseline pool is the paper’s full set: SASRec and HSTU, RecFormer fine-tuned and zero-shot, OneRec 8B Pro, four cheap rules (popularity, recent items, user-popular, and predict-last), and three candidate-bounded arms, Gemma 4 31B, LightGBM [64], and TabPFN v3 [43], which rank a shortlist of the training stream’s most frequent products (1,000 on most rows), a target outside it counting as a miss (Appendix F.4). Tables 8 and 9 describe each arm and which arm each ratio is against, Appendix F.1 sets out the conventions all grids share, and Table 7 profiles each dataset.

Where the margin comes from. Two design choices predict where the margin should be widest, and the grid follows both. First, the model reaches items through their multimodal features (Section 2.3), so an item absent from training is scored like any other. An atomic-ID specialist cannot reach it at all: zero-shot, the two specialists sit at chance on an unseen catalog, and even trained on the target’s split they rank only the items in its training interactions (8.6% of the catalog on reviews). The margin is accordingly widest where the evaluation window carries items the training window does not. Second, the model ranks the full catalog, with an exact softmax over 10^9 candidates (Section 2.5), so the margin grows with assortment. On the same stream, moving from a thousand items to the dataset’s own catalog leaves us $0.44\times$ of our accuracy and the candidate-bounded arms $0.21\text{--}0.23\times$ of theirs, roughly doubling the margin (Figure

	View recall 10 datasets	Full search 7 datasets	Partial search 6 datasets	Purchase recall 4 datasets	Item returned 2 datasets	Merchant 1 dataset	Category 1 dataset	
BehaviorGPT 12.5B fine-tuned	100	100	100	100	100	100	100	
BehaviorGPT 12.5B zero-shot	77	94	88	87	94	100	100	
BehaviorGPT 0.5B fine-tuned	78	94	94	89	75	78	75	
BehaviorGPT 0.5B zero-shot	50	89	83	50	60	78	75	
RecFormer	33	13	10	24	16	70	54	T
HSTU	34	q	q	28	3.2	45	11	T
LLM (Gemma 4 31B)	14	37	35	20	1.7	54	69	B
SASRec	29	q	q	28	3.8	13	11	T
user-popular	29	14	10	13	63	83	71	w
recent items	27	14	10	13	71	85	77	w
RecFormer zero-shot	17	2.4	5.9	6.3	4.7	66	40	w
OneRec 8B Pro	7.5	5.6	4.8	6.7	2.5	59	38	w
popularity	9.1	3.6	3.2	11	3.4	61	34	p
LTM (TabPFN v3)	2.5	q	q	3.4	1.4	68	46	B
gradient boosting (LightGBM)	1.7	q	q	1.4	0	51	8.4	B

Figure 10: **Which baseline is strongest depends on the task.** Figure 2 by task. A cell is the method’s median share of the leading method on that task, over the datasets that carry it (100 is the leader, 10 an order of magnitude behind); the header gives each column’s dataset count, from ten down to one. The two payments columns score hit@5 on a merchant or category rather than recall@10 on a product, and sit in the same grid because each cell is a share of its own column’s leader. Rows keep the pooled order, so a row read across is a method’s profile; marks follow Figure 2. **q** marks the four arms that never see the query (SASRec, HSTU, TabPFN, LightGBM); they are run on the search columns but excluded from the leader there, as in the breadth matrices.

23). Across datasets the margin correlates with catalog size on every arm (Spearman +0.34 to +0.73), though domain travels with size on this benchmark and only the two-point comparison is controlled (Figure 24; Appendix F.4).

Scale across tasks. Figure 12 replaces the baseline with our own 0.5B model on the same positions, so what varies is chiefly parameter count (chiefly, for the caveats below). Without target data (left panel), the 12.5B’s lead is uneven and grows with distance from the pretraining mixture: $0.97\text{--}1.42\times$ on retail, $4.4\times$ on short video, $7.2\text{--}9.1\times$ on image discovery, and up to $26\times$ on beverage discovery, a pattern set by what the model was pretrained on rather than by size alone. Two models support an ordering rather than a trend; that ordering is the behavioral analogue of the GPT-3 property from the introduction, and the form in which we state property (iii).

After adaptation (right panel), the 12.5B’s lead over the 0.5B narrows from $1.94\times$ to $1.47\times$ but persists. Every column narrows, and the rows furthest ahead fall furthest, beverage discovery from $13\times$ to $2.4\times$ and short video from $4.4\times$ to $1.4\times$. Target data substitutes for scale where the small model can reach the same ceiling, which is most tasks, but not for pretraining: a model trained from scratch on a target’s data levels off well short of one fine-tuned from a pretrained checkpoint. On beverage discovery, stage 2 from scratch levels off at 4.28 nats after twelve epochs, against 3.70 from a smaller pretrained checkpoint and 3.43 from our largest after four (Figure 26, Appendix G). Nowhere does scale cost accuracy: the four cells below parity, with or without target data, are all within three percent of level. Target data thus narrows the larger model’s lead but, on this grid, never reverses it; this is the form of property (iii) that matters for deployment.

Against the closest published comparison, OneRec 8B Pro run zero-shot on the same catalogs, the 12.5B leads on all 12 datasets, from $1.8\times$ on convenience to two orders of magnitude on beverage discovery. Two caveats bound the scale reading: two models fix a direction, not a curve (no power-law fit, no extrapolation past 12.5B), and the 0.5B was trained on the previous stage-2 pipeline and a different data mixture, so the ends of each row differ in more than parameter count.

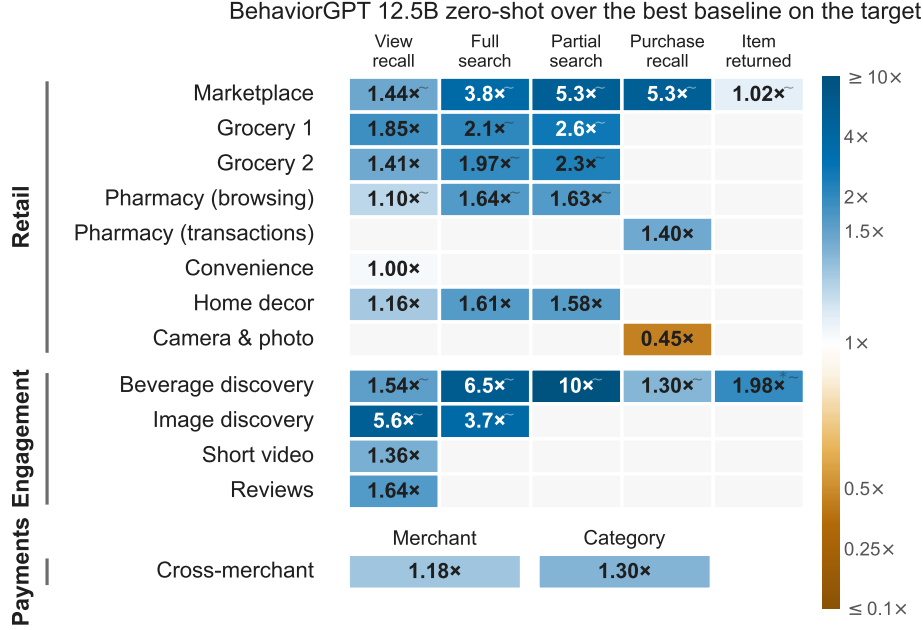


Figure 11: **Without training on the target, the same model leads the best available model on all but one cell of the grid.** Figure 5 redrawn for the zero-shot 12.5B, with the same rows, columns, and baseline rule. It leads 30 of the 31 cells, by a geometric mean of $1.95\times$; $\sim$ marks the 19 cells whose baseline scored a different set of positions, mostly fewer, where a run was cut or strided for cost (Appendix F.3). The margin is widest on search, at $3.1\times$ and $2.7\times$, where Gemma is the strongest baseline on eight of the 13 cells. It is narrowest on returns, at $1.42\times$, where a rule nearly matches us on marketplace (*recent items* 81.3% against our 82.9%, but 42.3% against 83.7% on beverage discovery), then after a purchase ($1.44\times$) and on a view ($1.57\times$). On the aggregate next-item slice that the literature benchmarks (not a column here), the margin is $1.8\times$. The one loss, camera & photo’s purchase recall at $0.45\times$, becomes a win after fine-tuning (Figure 5). *Beverage discovery’s return column holds only 98 scored positions and is directional only. Short video and reviews are scored under Figure 1’s protocol.

Accuracy at serving cost. Breadth bought with serving cost would not survive the constraints of Section 2.8, so Figure 3 sets each method’s leaderboard standing against what a ranked list costs it. The baselines occupy one band at every cost: SASRec and HSTU, at 0.18 and 0.26 ms per query, reach medians of 23% and 26%, and spending two orders of magnitude more, on Gemma 4 31B at 34 ms or OneRec 8B Pro at 20 ms, buys at most 32%. Our checkpoints sit above that band at the cost of the fast arms: the 0.5B at 82% fine-tuned and 69% zero-shot for 0.29 ms, and the 12.5B at 100% and 83% for 0.70 ms, its zero-shot point alone $2.6\times$ the best baseline’s median. The latencies fix an ordering rather than serving costs (Table 3), and every point of the upper band is one of ours.

3.3 Scaling Laws

A companion paper [37] reports a compute-optimal scaling study for this model family: roughly 600 runs from 10^{15} to 10^{19} training FLOPs, sweeping the embedder/contextualizer split, critical batch size, model/data allocation, and the number of sampled negatives after the embedder is frozen. Five findings shaped BehaviorGPT-v4’s configuration:

- **Compute accounting differs from LLMs.** The embedder runs on every event and candidate, so the per-step cost is $6BL(t p_e + p + 3BLh)$, with p_e and p the embedder and contextualizer parameters, $t \approx 24$ the embedder context in events, and the last term in-batch contrastive scoring. The embedder and the candidate axis, not the transformer alone, set the budget.
- **The compute-optimal embedder is small,** about 2% of parameters at every budget tested, because embedder parameters are touched far more often per step and see a far more repetitive data distribution (popular items recur thousands of times).
- **Allocation is data-heavy, and narrows toward Chinchilla with scale.** The optimal data-to-parameter ratio falls from ≈ 344 at 10^{15} FLOPs to ≈ 36 at 10^{19} ($N^* \propto C^{0.62}$): nearly an order of magnitude above the $\sim 20:1$ Chinchilla heuristic [47] over most of the range, and within about $2\times$ of it at the largest budget.

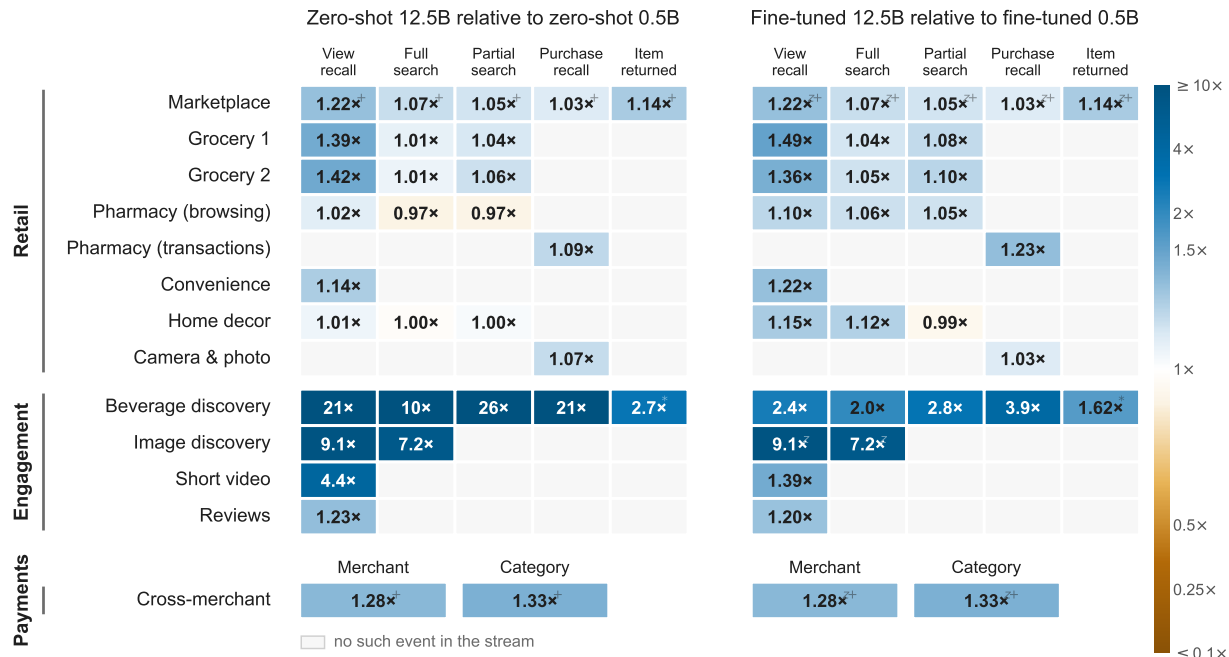


Figure 12: **What 25 $\times$ more parameters buy, and how much of it the target’s own data buys back.** The grid of Figure 5 with each cell the 12.5B’s recall@10 over the 0.5B’s on the same positions. *Left*: both zero-shot; the 12.5B improves on 28 of the 31 cells, by a geometric mean of 1.94 $\times$. *Right*: both fine-tuned on the same split where a fine-tune exists, and unadapted on the seven cells marked z ; the 12.5B leads 30 of the 31 cells by 1.47 $\times$, and 21 of the 22 that are its own fine-tune by 1.36 $\times$.

- **The evaluation metric is part of the scaling law.** Critical batch size, the optimal negative count, and how well the sampled-softmax loss tracks full-catalog ranking all shift with compute and with the target metric, so the training loss is not a dependable proxy for the deployed metric, and the metric must be chosen before the recipe is fixed.
- **Negative sampling becomes a memory question.** At 10^{19} FLOPs every headline metric still improves at the largest candidate pool trained, so candidate-axis memory, not FLOPs, is the binding constraint, which motivates the exact large-vocabulary softmax of Section 2.5.

What the law implies at this model’s size. Inverting the fitted allocation, $N^*(C) = 3.35 \times 10^{-4} C^{0.617}$ with $D^* = C/(6N^*)$, puts a 12.5B model’s compute-optimal budget at $C \approx 10^{22}$ FLOPs and its data at $D^* \approx 132$ B events, $D/N \approx 11$. BehaviorGPT-v4’s 150B actions over 12.5B non-embedding parameters give $D/N = 12$. This is consistency within a wide band, not confirmation: the extrapolation runs three decades past the largest sweep, the slope’s $\pm 1\sigma$ alone widens the optimum to roughly 1.5–90 events per parameter, and the slope leans on the largest budget, so fitting the four lower budgets alone gives an exponent of 0.58 and an optimum near 30. Nor can the form hold indefinitely: with an exponent above 0.5, $D^*/N^* \propto C^{-0.23}$ falls without limit, below one event per parameter near 10^{27} FLOPs, and we expect the exponent to flatten toward 0.5 as the optimum approaches the language-model regime. Measuring the frontier above 10^{19} FLOPs is future work. The embedder split lands the same way: 165M of 12.5B, or 1.3%, is inside the study’s optimal band (≈ 2 –3%, analytic optima [1.1%, 3.7%]) on its shallow side, where going below 2% costs little and going above 6% costs tenths of a nat. The 165M includes the embedder’s feature-vocabulary tables; its compute-active transformer alone would fall below the band, still in the shallow region.

Two caveats apply. The sweeps top out at 10^{19} FLOPs and optima of a few hundred million parameters, so at this scale the guidance is extrapolated rather than measured; and the allocation and architecture sweeps score against batch-local candidate pools, which keeps within-budget comparisons sound but makes absolute losses non-comparable across budgets. Figure 13 also places the published behavioral models of Table 1 against the frontier. The transferable recommenders sit one to two orders of magnitude below it, with too little data for even their few million parameters, consistent with their transfer resting on a frozen text encoder rather than on behavioral pretraining. Several industrial

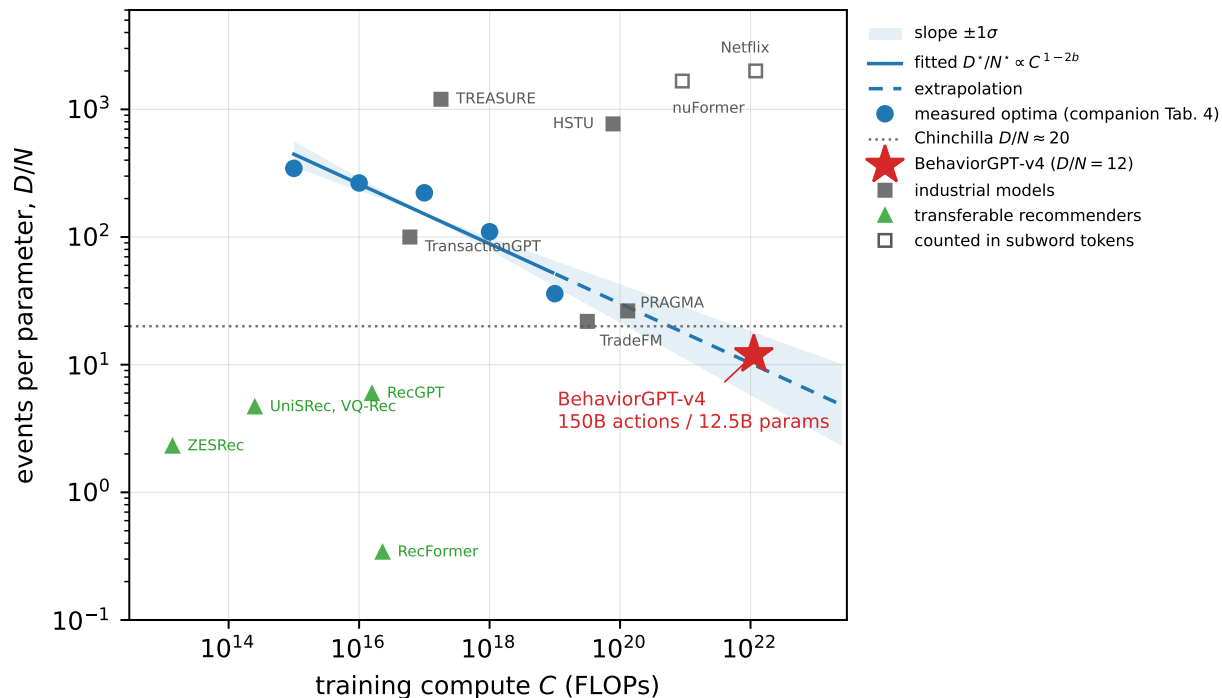


Figure 13: **BehaviorGPT-v4’s data-to-parameter allocation is consistent with the compute-optimal frontier of the companion scaling study; published behavioral models fall on either side of it.** The compute-optimal D^*/N^* of Gabrielsson et al. [37] (their Table 4, redrawn): five measured optima from 10^{15} to 10^{19} FLOPs, the fitted power law ($N^* \propto C^{0.617 \pm 0.025}$), and its extrapolation, the band being the slope’s $\pm 1\sigma$ alone. At BehaviorGPT-v4’s budget the extrapolated optimum is $D/N \approx 11$ events per parameter and the corpus supplies 12. The dotted rule is the text-LM Chinchilla heuristic. Grey squares and green triangles place the models of Table 1 at $C = 6ND$ from non-embedding counts (mostly our estimates) and reported corpus sizes. Their placement is indicative only: the frontier was fitted on our architecture and loss; D is corpus size, not data seen; hollow markers count subword tokens, several per event, which overstates D/N ; and academic sequence counts assume ten interactions per sequence.

models sit well above it, small for their data, most likely a trade for serving latency; if the frontier carries over to their architectures, a larger model could make use of that data. PRAGMA, TransactionGPT, and TradeFM sit near it.

3.4 Data Mixing

The pretraining mixture of Section 2.6 is fixed throughout this paper, so we report one mixing ablation at proxy scale, about two orders of magnitude below the full run. A base run sees 2B retail events (including grocery), and three arms see 8B: more retail data; the base data plus engagement data from image discovery under one shared softmax; and the same mixture with a separate softmax per distribution (Section 2.5). Appendix G.1 gives the full setup and its caveats, including that the separate-softmax arm is not a pure control.

Under separate softmaxes, the added image-discovery data improves on the base run in the mixture (-0.08 nats) and on all four catalogs outside it (Figure 14). At the same budget it trades a little in-domain accuracy for transfer: more retail data is better in the mixture (4.53 against 4.65 nats), while the image-discovery mix is better outside it (-0.20 against -0.05 nats relative to the base) and also covers image discovery itself, at 9.1% recall@10 on a stream the retail-only arms never saw. Under a shared softmax, the in-mixture stream ends 0.37 nats worse than the base run at a quarter of the compute, and neither a fivefold larger mixed corpus nor a larger model closes the gap. This is the experimental ground for the per-distribution softmaxes of Section 2.5.

3.5 Internal Experiments and Ablations

Appendix G reports in full the internal experiments and ablations that shaped the recipe; several bear directly on the paper’s claims. Events entering through learned feature embedders outperform one-hot identifiers on long-tail events, at

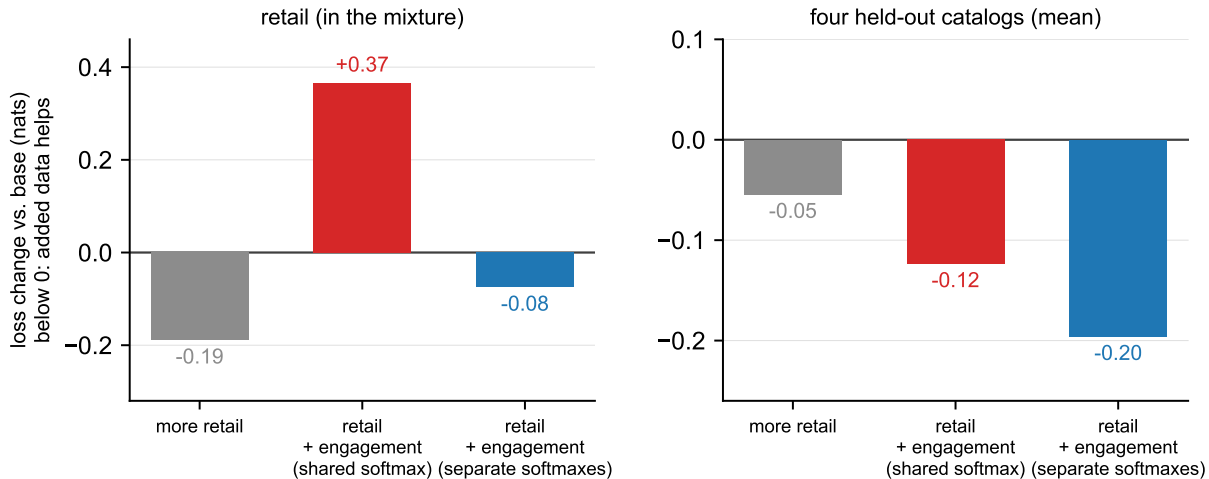


Figure 14: **Added out-of-industry data helps only with separate softmaxes.** Change in held-out loss when the training budget grows from a 2B-event retail base to 8B events, spent on more retail data or on added engagement data (image discovery). *Left:* the in-mixture retail stream. *Right:* mean next-product loss over four held-out catalogs (home decor, both pharmacies, camera & photo). Under one shared softmax, engagement data makes the in-mixture stream worse than the base run it was added to; under separate softmaxes it helps on both panels, and helps most outside the mixture.

equal data and at equal compute (Table 35), the experimental basis for features over IDs (Section 2.3). Image and text embedders trained end-to-end on behavior both outperform pretrained initializations (Tables 36 and 37), supporting the argument that our transfer runs through behavior rather than being inherited from language (Section 6.4). Pretraining on two grocery datasets improves one of them over pretraining on it alone, and removing domains that only one dataset carries improves it further (Table 31), since otherwise the model learns which dataset a sequence came from rather than the behavior they share. Design choices that favor in-domain fit can cost transfer: a softmax in both directions, as in CLIP, lowers in-domain loss but raises out-of-domain loss (Table 40), whereas separate softmaxes per candidate distribution improve both (Table 48), the basis of Sections 2.5 and 3.4.

3.6 Real-World Deployments (A/B Tests)

Offline metrics are an imperfect proxy for production outcomes. Table 4 summarizes multi-year A/B tests of BehaviorGPT deployments against incumbent commercial systems and internal baselines across search, recommendations, in-app engagement, physical retail, and assortment optimization; Appendix H sets out what the commercial comparisons can support and gives day-level results for the engagement test.

3.7 Qualitative Examples

Appendix I shows the storefront demo (Section 4) serving the 0.5B and 12.5B over a multimillion-item marketplace catalog, where every change in the ranking comes from the model’s input rather than from rules written for the demo. The ranking follows context (*chocolate* returns heart-shaped boxes on Valentine’s Day and gift boxes on Christmas Eve), earlier events (a Samsung phone in the cart brings up its case, charger, and earbuds), and both together (a baby-shower purchase and a late-October date produce baby Halloween costumes), and it reads queries against the session, resolving the misspelling *plystton* to PlayStation. A second demo, Curate my wall, ranks thousands of artworks from the Art Institute of Chicago, a catalog with no interaction data, and also suggests clothing: from a visitor’s art preferences alone, it recommends H&M products (Figure 35). A third demo detects fraud with no labels: it scores each action by how much it surprises the model, flags a session once the accumulated score crosses a threshold set on normal shoppers, and names the actions behind each flag, catching an account takeover one step after it begins (Figure 36). The model can also be queried rather than served: Appendix J.2 reads brand propensities off it, finding for instance that a broadband brand’s top products beyond the network itself are household staples and moving services, bought around a move.

Area	Metric/Outcome	Comparison Baseline	Gain
Recommendations (General)	Clickthrough rate (CTR)	Internal recommender	+29%
	Conversion rate	Most-bought-products baseline	10×
	Conversion rate	Visual art recommender engine	+24%
	Conversion rate, related products	Partner’s in-house related-products module	+20%
Engagement	Clicks on featured content	Control arm, niche-interest app	+11.7%
	Orders following a click	Control arm, niche-interest app	+19.5%
Search	Conversion rate	Commercial site-search vendor A	+5.7%
	Conversion rate	Commercial site-search vendor B	+7.5%
	Conversion rate	Commercial personalization vendor C	+9.4%
	Interactions with search-term suggestions	Incumbent suggestion mechanism	+49.1%
	Conversion rate	Commercial product-discovery vendor D	+16%
	Time on site (min:sec)	Vendor D (3:17 vs. 4:03)	+23%
	Search depth	Vendor D	+21%
	Custom uploads (lower is better)	Vendor D (4.41% vs. 1.29%)	3.4× fewer
	Search refinements	Vendor D	-20%
	Avg. order value	Vendor D	+5%
Physical Retail	In-store sales via dynamic assortment	Static regional assortment	+2.2%
Dynamic Categorization	Conversion rate	Incumbent categorization system	+11%
Assortment Optimization	Est. sales via gap filling	Manual assortment planning	+14%
Retraining Effects	Avg. conversion uplift (annual)	Previous model version	+2.5%

Table 4: **A/B tests of BehaviorGPT deployments, 2020–2026.** Each commercial row compares against one production deployment as configured by the partner (Section 3.6). Incumbents are anonymized as vendors A–D: A–C at one retail partner and D at another. Identities are available to reviewers on request, subject to partner consent.

4 Demos, SDK, and API

Claims about a foundation model can be verified only if others are able to use and evaluate it. Zero-shot transfer is also what makes it deployable: a catalog the model has never seen can be served as soon as it is uploaded, with no data collected and no model trained for it, and anything that consumes a ranking, a prediction, or an embedding can be built on top. We make the 12.5B and 0.5B checkpoints available through three access points:

1. **Live demos** at <https://www.unboxai.com/behaviorgpt>: an e-commerce storefront for exploring either checkpoint over deployed catalogs or a catalog the user uploads, an art-curation wall, and further applications built on the SDK (Appendix J.1).
2. **SDK and API** at <https://github.com/Unbox-AI/behaviorgpt>: deploy any catalog and call next-action prediction, ranking, retrieval, and embedding extraction on it, for search, recommendations, or any other application; we invite others to build their own.
3. **Example notebook** at <https://github.com/Unbox-AI/behaviorgpt>: runs our zero-shot PixelRec evaluation (Section 3.1) through the API. It illustrates the protocol rather than reproducing the paper: baselines are trained with their authors’ code, and results on internal data cannot be recomputed outside our infrastructure.

5 Discussion

A different kind of object. Zero-shot transfer and breadth from one checkpoint are capabilities a specialist lacks altogether: a model trained on one dataset has nothing to train on before the target dataset exists, and once it does, its weights serve only that dataset. That the same weights lead 30 of 31 dataset–task pairs without target data, each against a locally chosen opponent (Figure 11), and all 31 once adapted (Figure 5), thus argues for a foundation-model family of its own. It is distinct from LLMs, which it contains as a special case but which reach behavior only through descriptions of it, and from Large Tabular Models, which transfer a fitting procedure rather than knowledge of behavior; we outperform both on every benchmark task that can be posed to them.

Scale and target data. The case does not rest on the current model. From 0.5B to 12.5B, zero-shot transfer improves on 28 of 31 cells, and the gain survives adaptation at $1.47\times$ (Figure 12). Scale also substitutes for target data: on the two public datasets, the 0.5B model needs 10k–36k target samples to match the 12.5B model with none (Section 3.1), and nowhere on the grid does target data reverse the gap. Two sizes fix a direction, not a curve: we fit no power law, do not extrapolate past 12.5B, and note that the models also differ in pipeline and mixture.

Implications for behavior-driven industries. Every industrial model in Table 1 is pretrained and evaluated on a single platform. Our results suggest this is a limitation of the data, not of behavior: one model pretrained across three industries leads every specialist we compared. Out-of-industry data even improves catalogs outside the mixture, though only under separate softmaxes and at proxy scale (Section 3.4). This points to the path language took after GPT-3: rather than one model per domain and task, general behavioral models that adapt zero- or few-shot and are fine-tuned only where the data warrants it.

Complementary foundation models. LLMs and LBMs are trained on different data toward different ends: an LLM models how people describe the world, an LBM how they act in it. Their strengths are accordingly complementary. Gemma is the strongest search baseline because it reads the query, yet it can rank only a shortlist, whereas an LBM scores the full catalog in one pass (Table 3). We therefore expect the two to be used together, as in our most successful deployments: the LLM as language interface and source of knowledge, the LBM as the behavioral model it calls.

Beyond next-action tasks. All our tasks are next-action retrieval, but many behavioral decisions concern outcomes, such as whether a loan is repaid or a transaction is fraudulent. These can be handled by fitting a head, or embeddings from final or intermediate layers, on the target’s labels, or posed as next-action retrieval with outcomes embedded as candidates, keeping them to a single pass. The second option matters more: a consistent way to express every downstream task as behavior, which next-token prediction gave language and Section 2.2 provides here, lets a task inherit the zero-shot transfer of Section 3 instead of waiting for labels. Internally, a checkpoint’s next-action performance tracks its performance on credit underwriting and fraud detection; we defer these results, as regulated, high-stakes decisions warrant their own evaluation of fairness, calibration and explainability. In the workforce setting, next-event pretraining quality likewise correlated with attrition prediction, employee embeddings organize by tenure and reason for leaving, and a fine-tuned model reaches 91% accuracy [34]; earlier BehaviorGPT embeddings have similarly served beyond pretraining [36, 35]. As Figure 1 shows, pretraining changes what a target sample is worth, most where labels are scarce, as outcome labels usually are, arriving months after the behavior that predicts them.

What remains open. Three questions bound the claim. *Benchmark coverage:* public behavioral datasets are few and mostly retail recommendation, payments rests on one stream, and some cells have small evaluation sets. *Reasoning under a latency budget:* post-training for test-time compute in the action space (SFT on a reasoning format, then GRPO) nearly doubles next-purchase accuracy after a viewing session (13% to 25%), but in a modest-compute run lifts accuracy across tasks only from 8.9% to 9.7% (Table 52). Action traces are far shorter than language chains of thought, yet still forgo the single pass behavioral serving relies on (Table 3). Since the prompt selects reasoning within one checkpoint, it can be reserved for decisions worth the latency; distilling its gains into a single pass, or reasoning ahead of the request, remains open. *Beyond consumer behavior:* the same architecture and objective, trained separately on workforce events (a smaller event vocabulary, coarser time granularity), predict attrition [34] (Section 2.6). Whether such domains reinforce rather than dilute one another in one mixture is open; the mixing result suggests this depends as much on how the objective separates distributions as on which are included.

5.1 Limitations

The internal datasets behind most results cannot be shared, and the public ones are only proxies for deployment. Our scale evidence rests on two sizes, and the compute-optimal guidance we adopt [37] was measured well below the scale at which we apply it.

5.2 Ethics and Societal Impact

Behavioral modeling raises concerns distinct from language modeling: predicting human actions touches on privacy, consent, and power asymmetries between organizations and individuals. We follow established practice throughout, from data provenance, consent and de-identification to governance, fairness and responsible deployment.

6 Related Work

Our work draws on foundation models across language and other modalities (Section 6.1) and on recommendation systems (Section 6.2); Section 6.4 compares it with the foundation models already built on behavioral data, and Section 6.5 with LLM-based agents and reasoning for behavior.

Our line of work. We have pursued foundation models for behavior since 2020, when we proposed modeling behavior, transactions and retail with language-model techniques [12], and the following year formulated a data and model design general across behavioral domains [13]. In 2023 we presented the approach in an MIT lecture series [14]. In 2025 we introduced BehaviorGPT [36], applied it to workforce dynamics and visual aesthetics [34, 35], and presented it in a series of talks [16, 15, 17]; in 2026 we reported scaling laws for behavioral foundation models [37] and proposed a definition of Large Behavioral Models in a position paper [33]. The majority of the works reviewed below appeared over the same period.

6.1 Foundation Models

From language to behavior. An early challenge for foundation models was framing language tasks so that one model could generalize across many of them. Word2vec [79] learned meaning from co-occurrence, after the distributional hypothesis [31], much as an LBM defines an entity by its co-occurrence in sequences of actions (Section 1). Word embeddings inspired the pretrain-then-fine-tune paradigm popularized by BERT [26], which we use when target data is available. Casting NLP tasks as one sequence problem [77] paved the way for autoregressive models that, at scale, learn in context from few examples or none [11]. Next-token pretraining followed by instruction tuning, RLHF [84] or GRPO [25] remains the dominant recipe, with prompting conditioning the model on a task. We extended these developments to behavior [36, 34, 35] and found that many transfer well: autoregressive causal training proved most effective, framing every task as generation remains the most effective approach we know of and lets one checkpoint serve many tasks, and GRPO carries over to reasoning in the action space (Appendix B.8). We also observed early on that many language tasks have direct behavioral analogues: translation corresponds to cross-selling, sentiment analysis to mood inference, and text generation to recommendation or search (Table 5). Transfer has also run the other way, with relative position prediction, an objective developed for behavior, improving text encoders [18]. Behavior departs from language in that predicting backward, such as which search most likely preceded a purchase, is often as useful as predicting forward, which calls for a more nuanced use of autoregression (Appendix B.5).

Language task	Behavior analogue	Illustrative application
Machine translation	Cross-selling	Map grocery baskets to electronics suggestions
Sentiment analysis	User-mood inference	Predict satisfaction from purchase streams
Question answering	Goal-conditioned retrieval	Fetch products given query-like contexts
Text generation	Action recommendation	Autoregressively suggest next item(s)
Sequence classification	Behavioral risk scoring	Flag churn or fraud from traces

Table 5: Task analogies between language and behavior domains.

Foundation models beyond language. Self-supervised pretraining [6, 5, 20] has spread to genomics [111], electronic health records [104], consumer behavior [36] and time series [24, 103]. These efforts share the pretrain-then-fine-tune paradigm and largely adopt autoregressive modeling, but each domain’s characteristics have called for bespoke techniques. Large Tabular Models extend the recipe to tables, transferring through synthetic priors or schema text [48, 45]; we treat them as a baseline family throughout and compare them with our setting in Section 6.4. Four such characteristics also shape behavioral modeling. *Format*: unlike images and text, time series are heterogeneous, interleave diverse event types, and are shaped by their sampling frequency, and behavioral data shares these issues. *Interpretability*: like DNA, behavior is less structured and interpretable than language, which makes prompting less intuitive. *Vocabulary*: four bases in genomics, hundreds of thousands of tokens in LLMs, and billions of candidate items in recommendation create very different bottlenecks. *Cost*: finance, payments and many of our use cases need inference far faster and cheaper than current LLMs provide, because their applications are high-frequency and low-margin.

Vision foundation models and the role of behavior. Vision foundation models [90, 4] learn largely from static images and their captions. Robotics work points to a limit of that data: models trained on static images or text can misread scenes, and training on visual traces of behavior helps, because elements are understood through their role in an interaction rather than their appearance [114]. LBMs rest on the same grounding, defining meaning through patterns

in user sequences and so surfacing preferences and intent that pixels miss, whereas datasets of what people say they see, such as art descriptions [57, 8], record conscious, narrow interpretations rather than behavior. LBMs are thus built to complement vision and language models rather than replace them.

6.2 Recommendation Systems

Recommendation was long dominated by collaborative filtering. Among the earliest deep approaches, YouTube’s two-tower model [23] encodes users (demographics and behavior) and items with separate networks and retrieves by the inner product of the two. Sequential modeling followed, drawing on NLP: GRU4Rec [46] pioneered recurrent session-based recommendation, BERT4Rec [96] mirrored BERT’s masked-language objective to learn bidirectional representations of interaction sequences, causal Transformers proved highly effective [62], and LLMs have more recently entered the field [106].

Behavioral modeling across domains and time. Our setting spans many behavioral domains, so the goal is to generalize across distributions rather than model one. Multi-domain recommendation trains one model across a platform’s scenarios with shared and domain-specific parameters [93, 76, 97], and cross-domain recommendation transfers from a source to a target domain, almost always through shared users or items [118]. Both stay within one platform or depend on overlap, and neither studies how the candidate space should be divided when domains are pooled, which our mixing ablation finds decides whether added data helps or hurts (Section 3.4). Learning across industries with disjoint catalogs and users has received little study [30]: we want grocery purchases to inform behavior in electronics, and the model to adapt to any reasonable behavioral space, such as a new or shifting assortment.

Time is another form of domain shift. Lexical drift in language is slow, but consumption can shift within weeks as new trends and products appear: the launch of an iPhone branded with “Apple Intelligence” may immediately change preferences and actions. Treating temporal dynamics as domain change unifies how we think about generalization, and augmentations, including contrastive and dropout-based strategies, encode invariances that improve robustness to both kinds of shift [20, 32, 95].

A unified formulation. HSTU [112] models app behavior with separate retrieval and ranking tasks:

$$i_1, a_1, i_2, a_2, \dots, i_n, a_n \rightarrow i_{n+1} \quad \text{retrieval} \quad (2)$$

$$i_1, a_1, i_2, a_2, \dots, i_n, a_n, i_{n+1} \rightarrow a_{n+1} \quad \text{ranking} \quad (3)$$

whereas we use one format:

$$a_1, i_1, a_2, i_2 \dots, a_n, i_n, a_{n+1} \rightarrow i_{n+1} \quad \text{retrieval \& ranking} \quad (4)$$

Here a_n can refer to the upcoming or the preceding item: the task is retrieval in the first case and ranking in the second, and training both jointly yields a model that generalizes across behavioral domains and interaction types. Unified formulations have been explored for end-to-end training across recommendation tasks [41], but, as in language modeling [91, 11], we find autoregressive causal training the most efficient, with many of its benefits transferring to behavior.

Generative recommenders and quantization. The size of the candidate set is a central challenge in recommendation. Generative recommenders with learned quantization [92], built on residual quantization [71], address it by representing each item as a short code: three tokens of 1,000 values each give $1000^3 = 10^9$ combinations, so a billion items need three indices each and 3,000 shared embeddings rather than a billion. The memory saving costs complexity in training and inference: decoding typically requires beam search over longer sequences, turning a single scoring step into tens of sequential passes (Table 3). In our long experience with such methods, efficient GPU and CUDA optimization has handled large vocabularies better while keeping inference latency low at scale, though we still use learned quantization to ease memory constraints without sacrificing performance.

6.3 Large Behavior Models in Robotics

Parallel work uses similar names for short-horizon motor tasks: Large Behavior Models for multitask dexterous manipulation [99], and, unrelated to ours, BehaviorGPT for simulating agents in autonomous driving [116]. We focus instead on discrete, long-horizon domains such as retail, payments and engagement, where actions unfold over months or years. One can imagine a unified model that both controls a household robot and manages its owner’s shopping. Either way, the term as we use it (Section 1) requires that the model read sequences of human actions in chronological order, rather than abstractions, artifacts or reinterpretations of behavior; Appendix A gives the extended argument.

6.4 Behavioral Foundation Models

Industrial event-stream models. The closest work applies foundation-model training to large industrial event streams; over the past two years, media, commerce, banking, card networks, payments and markets have each published such a model. In engagement, HSTU [112] recasts ranking and retrieval as generative sequential transduction on Meta’s platforms, and Netflix reports a pretrained recommendation backbone that improves with scale [82, 107]. In retail, Shopify’s Commerce Foundation Models run HSTU-based models over buyer and merchant event streams [3, 67]. In finance, PRAGMA pretrains on 24 billion Revolut banking events [83], nuFormer on about 500 billion tokens of Nubank transactions [10], Visa has published TREASURE [109] and TransactionGPT [29], Stripe reports a payments foundation model [65], and J.P. Morgan’s TradeFM models trade flow [63]. Wukong [113] and Ardalani et al. [2] show that recommendation architectures scale predictably. Together, these establish that within one platform a pretrained sequence model can replace hand-engineered features and per-task pipelines at production scale.

They leave open whether behavioral pretraining is a *foundation* in the sense the term has in language, through three recurring gaps. **Single-domain pretraining:** each corpus is one company’s stream, so cross-domain transfer is never tested. **Almost no zero-shot regime:** downstream results come from fitting a head, probe or adapter on target labels, with the pretrained model as a feature extractor. The exception, TradeFM, transfers zero-shot from US to Japanese and Chinese equities because its inputs carry no instrument or venue identifier, our point about ID-based representations reached from the other direction, though within one asset class rather than across catalogs or industries. **Proprietary evaluation:** results are relative lifts on proprietary data, which limits external comparison; HSTU does report public benchmarks, but trains a model per dataset from scratch, the specialist setting rather than transfer.

Transferable recommenders. A second, largely disjoint literature targets transfer directly and reports zero-shot results. Because item text is shared across platforms where item IDs are not, ZESRec [27], UniSRec [50], VQ-Rec [51] and RecFormer [72] represent items by their descriptions, pretrain on a mixture of public datasets and evaluate on held-out ones; RecGPT [60] extends this to a decoder-only generative recommender that beats baselines trained on 10–50% of the target data, and LLMs prompted as rankers [53] offer a further zero-shot baseline. We therefore do not claim zero-shot behavioral transfer as new, only the regime in which it holds. This literature pretrains on 10^6 – 10^7 sequences from public recommendation corpora of one kind, so “cross-domain” means Amazon Scientific to Amazon Instruments rather than grocery baskets to card transactions to in-app sessions. Its results are typically reported under leave-one-out splits with sampled negatives [69, 59] against baselines trained on similarly small data, and its target catalogs hold at most 4×10^4 items [72, 92], against our 2×10^7 (Table 7).

Generative recommenders on language models. The closest single comparison is OneRec-Foundation [115]: open-sourced, covering eight task families across short video, advertising and e-commerce, and gaining 26.8% average Recall@10 over the strongest baseline on ten held-out Amazon datasets. It differs from our setting in three ways. Its transfer is not zero-shot: the Amazon results fine-tune on the target under leave-one-out splits, whereas we use no target samples and rank the full catalog under temporal splits. Its corpus is one company’s surfaces at about 10^{11} tokens, over an order of magnitude below ours. And its generalization runs through language: the backbone is Qwen3, its pretrained itemic tokenizer collides on over 30% of Amazon items, and the authors’ best remedy replaces behavioral tokens with five text keywords per item, consistent with the limits we expect when similarity is derived from language. We read this line as complementary, showing how much can be recovered by mounting recommendation on a language model when behavioral data is limited to one platform.

Large Tabular Models. A third literature appears to have what we claim already. TabPFN and its successors [48, 49], trained on synthetic priors in the Neural Processes tradition [39] and extended to time series by TimePFN [28], together with cross-table and language-serialized variants [117, 68, 45, 38], predict on unseen datasets without gradient steps and are domain-agnostic by construction, at least as broad as we are across industries. But they are not behavioral, and three limits show in our results. An LTM consumes a fixed-width table of i.i.d. rows, so an event history must be flattened into hand-engineered aggregates, discarding order and reintroducing the feature engineering behavioral pretraining exists to remove. Its context is bounded at a few thousand rows, which is why the in-context methods in Figure 1 lead at the smallest budgets and then saturate. And it predicts a label from a small fixed set, so ranking an open catalog cannot be posed to it at all. What transfers also differs: TabPFN’s prior over synthetic causal graphs holds no knowledge of what people buy, and none could be written for behavior, whose patterns are culturally specific, non-stationary and spread over billions of actions; augmentations [32] play the role synthetic data plays there. LTMs have learned *how to fit* a small dataset quickly; an LBM has learned *what people do*. We evaluate against them wherever the formulation permits, as the strongest zero-shot alternative on the tasks both can pose.

Carrier and geometry. Scale aside, these lines differ in *how* transfer is achieved, which Table 1 separates into two columns. The *carrier* is the representation through which an event enters the model, and so the only thing that can

reach an unseen catalog: *IDs* cannot leave their platform, since a new catalog has no rows in the embedding table; *schema* fields travel only between platforms that share a schema; *text* travels wherever language does; and a synthetic *prior* travels anywhere. The *geometry* is what makes two events close once inside. Every industrial model learns a behavioral geometry; we differ in pairing it with a carrier that survives leaving the platform. The industrial models bind their carrier to their catalog, and say so: HSTU reports that a content-only variant substantially underperforms its ID-based model, TREASURE maps new merchants into shared aggregate buckets, and Netflix’s content path, which mixes a title-metadata embedding with an ID embedding according to title age [82], is designed to cold-start titles within its own catalog and defers to identity once interaction data exists. TradeFM, which transfers because it drops identity entirely, suggests that the carrier, not the domain, sets the ceiling. The models that do reach a new dataset take their geometry from language: BERT encodes descriptions in ZESRec and UniSRec, a language model feeds VQ-Rec’s quantizer, RecFormer models attribute strings, and MPNet supplies RecGPT’s item embeddings. In three of these the encoder is frozen, so the pathway carrying the transfer is never trained on behavior, and their zero-shot capability is largely *inherited from language* rather than learned from behavior.

Our design principle rejects that geometry: regular and lactose-free milk are near-duplicates in description and nearly disjoint in behavior (Section 1). Linguistic transfer should therefore degrade where item text is uninformative, as with short search terms, brand tokens and merchant strings, and on such data pretrained text embedders *underperform* embedders trained end-to-end on behavior (Section 3). PixelRec, cover imagery with sparse text, is the sharpest public instance (Section 3.1).

The gap we address. Our claim sits at the intersection these literatures leave empty. The industrial models have scale, task breadth and deployment, but operate on one platform and, apart from TradeFM, never test transfer. The transferable recommenders test transfer across catalogs, but for one task and at a scale and domain diversity that cannot tell whether behavioral pretraining is a foundation-model story or a text-encoder story. OneRec-Foundation reaches new datasets only through a language backbone and after fine-tuning on the target. The tabular models apply to any domain and need only the target’s rows in context, but cannot rank even moderately sized catalogs, read images, or represent user sequences well, and transfer a fitting procedure rather than knowledge of behavior. BehaviorGPT-v4 is pretrained on 150 billion actions spanning three industries, served as one checkpoint across their subdomains and task families, transfers through multimodal behavioral representations rather than item text alone, is evaluated zero-shot against *fully trained* specialists under full-catalog ranking with temporal splits, and is deployed in production (Table 1). In non-embedding parameters, it is more than ten times the largest industrial behavioral model whose count can be estimated from the public record, and four to five orders of magnitude above the transferable recommenders; its corpus is also the largest disclosed, 150B actions against HSTU’s 100B examples and 3.6T tokens against the 2T per training cycle Netflix reports (Appendix A.1). Both comparisons concern the public record rather than what exists. Size matters here for what it lets us test: transfer at 10^4 – 10^5 times the pretraining scale of prior transfer studies, and transfer across industries, which to our knowledge has not been tested at any scale.

6.5 LLMs, Agents, and Reasoning for Behavior

Building on the success of LLMs, recent work uses them for recommendation more natively [73]: a prompt built from user history asks the LLM to describe the next likely action or product, and a retrieval model matches the description to items, so results still depend heavily on that retrieval model. Assistants and agents for recommendation, consumption and behavioral tasks [100, 105, 56] likewise reach user behavior and product spaces through retrieval modules (RAG), prompt chaining and tool use, with the LLM as a flexible orchestrator. We find this one-sided interaction limiting, since the retriever only retrieves and the LLM must supply contextual and behavioral intelligence on its own. We instead propose a two-sided interaction in which the LLM also communicates with the LBM: beyond retrieving products, BehaviorGPT can surface the behavior that precedes an action (Appendix J.2), group results into personalized themes (Appendix I) and generate keywords and search terms [35], giving the LLM behavioral context it cannot infer on its own.

Advances in prompting [101], RLHF [84] and reinforcement learning [25] have spurred work on LLM reasoning and test-time scaling [94]. Training LLMs end-to-end with tools such as web search and browsing is of growing interest, but is bottlenecked by tool latency and by LLMs not being pretrained on behavioral data, so they often reason about behavior in abstract language, ungrounded in the action space. Reasoning need not leave the space the model acts in: in BehaviorGPT it happens within the behavioral space, where searching, clicking and selecting are native actions rather than external tools. Independently, ReaRec [98] spends test-time compute inside a sequential recommender by feeding its hidden state back for several latent steps; our reasoning trace is instead a sequence of generated actions, trained with GRPO against the realized next action (Appendix B.8). Each step is a single action, costing about one LLM token, and a trace typically needs fewer steps than a chain of thought in words, so the reasoning is personalized, grounded in the action space, and faster than reasoning about behavior in language.

7 Conclusion

We presented BehaviorGPT-v4, a 12.5B-parameter Large Behavioral Model trained by next-event prediction on 150 billion user actions, and showed evidence that it has the three properties that made language models foundation models. It transfers zero-shot: on public benchmarks whose catalogs and users it has never seen, it outperforms baselines trained on those datasets in full, at every data budget we tried. It is broad: without target data, one checkpoint with no per-dataset changes leads the strongest available specialist on 30 of 31 dataset–task pairs across three industries, and on all 31 once adapted. And its generality grows with scale: from 0.5B to 12.5B, zero-shot transfer improves on 28 of 31 pairs, and the target data needed for a given accuracy falls. In production A/B tests under live-serving constraints, BehaviorGPT models raised conversion and orders by double-digit percentages against incumbent commercial systems in several deployments. Open questions remain, from wider benchmarks to reasoning within a latency budget and domains beyond consumer behavior. Still, our results suggest that behavior-driven industries may follow the path language took: toward general behavioral models that transfer to new domains and tasks zero-shot or from a few samples, rather than a separate model for every one.

Acknowledgments

We thank Ali Ghodsi, Naveen Rao, Maximilian Titz, Andreas Källström Säfweräng, and Harald Mix for their support, and Artem Sorokin for helpful discussions. We thank Alexander Hagberg for his work on the API, the SDK, and the storefront demo; Alexandros Lemonaris for his help with Figure 26; Johannes Schaffrath for his help with Table 30; and Felix Goetze for his early support. We are grateful to all our current and former Unbox AI colleagues. Google Cloud’s infrastructure and tools were instrumental in training and evaluating the BehaviorGPT models. Finally, we thank our partners and clients, from Fortune 50 companies to small and medium-sized businesses across retail, engagement, and payments, for their collaboration, domain expertise, and feedback, without which this work would not have been possible.

Contributions

Rickard Brüel-Gabrielsson conceived and led the project, designed the model, training objectives, and methodology, implemented the first versions of the system and ran all early experiments, directed the training of the BehaviorGPT models, including their scaling, data mix, and evaluation protocol, and wrote the paper.

Vasudev Gupta worked on the training codebase, model training, and evaluations over several years during the middle phases of the project, and gave feedback on the writing.

Tom Boustedt and **Simon Granström** ran the BehaviorGPT-v4 training runs, including optimizing them for NVIDIA B200 GPUs, carried out the breadth evaluation across datasets and tasks, and gave feedback on the writing.

Adam Fredriksson helped with training BehaviorGPT-v4, carried out the zero-shot and fine-tuning experiments on public benchmarks, and gave feedback on the writing.

Marcel Rød advised on GPU and tokenization optimization for training.

Gonçalo Marques led the development of the API through which the models, demos, and evaluation harness are served.

Jens Palmborg helped secure resources and domain experts, helped enable several of the production A/B tests in the paper, and gave feedback on the writing.

Erik Guander worked on the storefront demo, helped secure domain experts and resources, and gave feedback on the writing.

Nicolas Sanchez and **Artem Lukoianov** contributed through many discussions throughout the project.

Marc Mathieu and **Miguel Paredes** advised on the project.

Srijan Sood, **Georgios Kolovos**, **Simon Ejdemyr**, **Toni Rosino**, **Mattias Holmström**, **Pratik Thakar**, **Wen Yao**, **Peter Sarlin**, **Alexander Statnikov**, **Steve Flinter**, **Sebastian Siemiatkowski**, and **Bryan McCann** advised on the project in a personal capacity.

John Brüel-Gabrielsson provided advice throughout the project.

Gunnar Carlsson provided high-level advice during the first years of the project.

References

- [1] Diogo Almeida. Introducing System One Models & Jev. TypeSafe AI blog, September 15, 2026. <https://typesafe.ai/blog/introducing-system-one-models-and-jev>, 2026.
- [2] Newsha Ardalani, Carole-Jean Wu, Zeliang Chen, Bhargav Bhushanam, and Adnan Aziz. Understanding scaling laws for recommendation models. *arXiv preprint arXiv:2208.08489*, 2022. URL <https://arxiv.org/abs/2208.08489>.
- [3] Diego Ardila, Yang Liu, and Andrew Briand. Building shopify’s commerce foundation models. NVIDIA GTC San Jose, session S81636, March 2026. URL <https://www.nvidia.com/en-us/on-demand/session/gtc26-s81636/>. Accessed 2026.
- [4] Muhammad Awais, Muzammal Naseer, Salman Khan, Rao Muhammad Anwer, Hisham Cholakkal, Mubarak Shah, Ming-Hsuan Yang, and Fahad Shahbaz Khan. Foundational models defining a new era in vision: A survey and outlook. *arXiv preprint arXiv:2307.13721*, 2023. URL <https://arxiv.org/abs/2307.13721>.

- [5] Alexei Baevski, Wei-Ning Hsu, Qiantong Xu, Arun Babu, Jiatao Gu, and Michael Auli. data2vec: A general framework for self-supervised learning in speech, vision and language. In *International Conference on Machine Learning (ICML)*, 2022. URL <https://arxiv.org/abs/2202.03555>.
- [6] Adrien Bardes, Jean Ponce, and Yann LeCun. Vicreg: Variance-invariance-covariance regularization for self-supervised learning. In *International Conference on Learning Representations (ICLR)*, 2022. URL <https://arxiv.org/abs/2105.04906>.
- [7] Yoshua Bengio and Jean-Sébastien Senécal. Adaptive importance sampling to accelerate training of a neural probabilistic language model. *IEEE Transactions on Neural Networks*, 19(4):713–722, 2008. doi: 10.1109/TNN.2007.912312.
- [8] Yi Bin, Wenhao Shi, Yujuan Ding, Zhiqiang Hu, Zheng Wang, Yang Yang, See-Kiong Ng, and Heng Tao Shen. Gallerygpt: Analyzing paintings with large multimodal models. In *Proceedings of the 32nd ACM International Conference on Multimedia*, MM ’24, page 7734–7743. ACM, October 2024. doi: 10.1145/3664647.3681656. URL <http://dx.doi.org/10.1145/3664647.3681656>.
- [9] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Dombouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kudipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Muniyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2022. URL <https://arxiv.org/abs/2108.07258>.
- [10] D. T. Braithwaite, Misael Cavalcanti, R. Austin McEver, Hiroto Udagawa, Daniel Silva, Rohan Ramanath, Felipe Meneses, Arissa Yoshida, Evan Wingert, Matheus Ramos, Brian Zanfelicce, and Aman Gupta. Your spending needs attention: Modeling financial habits with transformers. *arXiv preprint arXiv:2507.23267*, 2026. URL <https://arxiv.org/abs/2507.23267>. KDD 9th Workshop on Machine Learning in Finance.
- [11] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfc4967418bfb8ac142f64a-Paper.pdf.
- [12] Rickard Brüel Gabrielsson. Unbox AI: Modelling behaviour, transactions, and retail with language-model techniques. Unbox AI pitch deck dated October 30, 2020 (redacted), released May 2025. <https://research.unboxai.com/figures/2020-10-30-UnboxAI-deck-redacted.pdf>, 2020.
- [13] Rickard Brüel Gabrielsson. Data and model sketch. Unbox AI design document dated June 22, 2021, released May 2025. <https://research.unboxai.com/figures/2021-06-22-Data-and-Model-Sketch.pdf>, 2021.
- [14] Rickard Brüel Gabrielsson. Future of AI is foundation models & self-supervised learning – MIT lecture 1. Lecture in *Future of AI: Self-Supervised Learning and World Models*, MIT EECS Independent Activities Period (IAP), January 12–February 2, 2023. Recording posted February 5, 2023. <https://www.youtube.com/watch?v=R6ITDsKX2c0&t=2725s>, 2023. Foundation models for behavior and retail discussed from 45:25.

- [15] Rickard Brüel Gabrielsson. Generative AI for behavior, actions, and transactions: Unbox AI. Talk, MIT Industrial Liaison Program, April 2025. <https://ilp.mit.edu/watch/generative-ai-behavior-actions-and-transactions-unbox-ai>, 2025.
- [16] Rickard Brüel Gabrielsson. BehaviorGPT: Large behavioral models. Talk at the MIT Sloan AI & ML Conference, February 2025. <https://www.youtube.com/watch?v=oq5Y1XB85XA>, 2025.
- [17] Rickard Brüel Gabrielsson. Actions speak louder than LLMs (behavioral AI). TEDxMIT talk, recording posted July 29, 2025. <https://www.youtube.com/watch?v=oj40vQnFaXs>, 2025.
- [18] Rickard Brüel-Gabrielsson and Chris Scovel. Relative position prediction as pre-training for text encoders. *arXiv preprint arXiv:2202.01145*, 2022. URL <https://arxiv.org/abs/2202.01145>.
- [19] Rickard Brüel-Gabrielsson, Jiacheng Zhu, Onkar Bhardwaj, Leshem Choshen, Kristjan Greenewald, Mikhail Yurochkin, and Justin Solomon. Compress then serve: Serving thousands of lora adapters with little overhead. In *International Conference on Machine Learning (ICML)*, 2025. URL <https://arxiv.org/abs/2407.00066>.
- [20] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning (ICML)*, 2020. URL <https://arxiv.org/abs/2002.05709>.
- [21] Yu Cheng, Yunzhu Pan, Jiaqi Zhang, Yongxin Ni, Aixin Sun, and Fajie Yuan. An image dataset for benchmarking recommender systems with raw pixels. In *Proceedings of the 2024 SIAM International Conference on Data Mining (SDM)*, pages 418–426, 2024. doi: 10.1137/1.9781611978032.49.
- [22] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 8440–8451, 2020. URL <https://arxiv.org/abs/1911.02116>.
- [23] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys)*, Boston, MA, USA, 2016. ACM, ACM. URL <https://storage.googleapis.com/pub-tools-public-publication-data/pdf/45530.pdf>. Google, Mountain View, CA.
- [24] Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting. In *International Conference on Machine Learning (ICML)*, 2024. URL <https://arxiv.org/abs/2310.10688>.
- [25] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaoqun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangan Zhou, Shanhuan Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shutong Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanbiao Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yudian Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song,

- Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Nature*, 645:633–638, 2025. doi: 10.1038/s41586-025-09422-z. URL <https://arxiv.org/abs/2501.12948>.
- [26] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL <https://aclanthology.org/N19-1423/>.
 - [27] Hao Ding, Yifei Ma, Anoop Deoras, Yuyang Wang, and Hao Wang. Zero-shot recommender systems. *arXiv preprint arXiv:2105.08318*, 2021. URL <https://arxiv.org/abs/2105.08318>.
 - [28] Samuel Dooley, Gurnoor Singh Khurana, Chirag Mohapatra, Siddhartha V Naidu, and Colin White. Timepfm: Effective multivariate time series forecasting with synthetic data. *arXiv preprint arXiv:2501.04696*, 2024. URL <https://arxiv.org/abs/2501.04696>.
 - [29] Yingtong Dou, Zhimeng Jiang, Tianyi Zhang, Mingzhi Hu, Zhichao Xu, et al. TransactionGPT. *arXiv preprint arXiv:2511.08939*, 2025. URL <https://arxiv.org/abs/2511.08939>.
 - [30] Kim Falk and Chen Karako. Optimizing product recommendations for millions of merchants. In *Proceedings of the 16th ACM Conference on Recommender Systems, RecSys '22*, page 499–501, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392785. doi: 10.1145/3523227.3547393. URL <https://doi.org/10.1145/3523227.3547393>.
 - [31] J. R. Firth. *Papers in Linguistics 1934–1951*. Oxford University Press, London, 1957. "You shall know a word by the company it keeps."
 - [32] Rickard Br  l Gabrielsson, Tongzhou Wang, Manel Baradad, and Justin Solomon. Deep augmentation: Dropout as augmentation for self-supervised learning. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL <https://openreview.net/forum?id=0jWB2671AR>.
 - [33] Rickard Br  l Gabrielsson. Large behavioral models: A foundation model paradigm for human actions. *Unbox AI Research*, 2026. URL <https://research.unboxai.com/large-behavioral-models.pdf>.
 - [34] Rickard Br  l Gabrielsson, Vasudev Gupta, et al. Behaviorgpt at work: A foundation model for workforce actions & dynamics through large behavioral modeling. *Unbox AI Blog*, jun 2025. URL <https://research.unboxai.com/behaviorgpt-foundation-model-workforce>.
 - [35] Rickard Br  l Gabrielsson, Vasudev Gupta, et al. Behaviorgpt for visual art: A foundation model for aesthetics. *Unbox AI Blog*, jul 2025. URL <https://research.unboxai.com/behaviorgpt-visual-art-and-aesthetics.html>.
 - [36] Rickard Br  l Gabrielsson et al. A foundation model for consumption, transactions, and actions: The inception of behaviorgpt. *Unbox AI Blog*, may 2025. URL <https://research.unboxai.com/foundation-model-for-consumption-transactions-and-actions.html>.
 - [37] Rickard Br  l Gabrielsson et al. Scaling laws for behavioral foundation models over user event sequences. *arXiv preprint arXiv:2606.05257*, 2026. URL <https://arxiv.org/abs/2606.05257>. arXiv:2606.05257.
 - [38] Josh Gardner, Juan C. Perdomo, and Ludwig Schmidt. Large scale transfer learning for tabular data via language modeling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL <https://arxiv.org/abs/2406.12031>.
 - [39] Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, S M Ali Eslami, and Yee Whye Teh. Neural processes. In *ICML Workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018. URL <https://arxiv.org/abs/1807.01622>.
 - [40] Gemma Team. Gemma 4 model card. Google DeepMind, 2026. https://ai.google.dev/gemma/docs/core/model_card_4, 2026.

- [41] Shijie Geng, Shuchang Liu, Zuohui Fu, Yingqiang Ge, and Yongfeng Zhang. Recommendation as language processing (rlp): A unified pretrain, personalized prompt & predict paradigm (p5). In *Proceedings of the 16th ACM Conference on Recommender Systems (RecSys)*, 2022. doi: 10.1145/3523227.3546767. URL <https://arxiv.org/abs/2203.13366>.
- [42] Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. In *International Conference on Machine Learning (ICML)*, 2024. URL <https://arxiv.org/abs/2404.19737>.
- [43] Léo Grinsztajn, Klemens Flöge, Oscar Key, Felix Birkel, Philipp Jund, et al. TabPFN-3: Technical report. *arXiv preprint arXiv:2605.13986*, 2026.
- [44] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9726–9735, 2020. doi: 10.1109/CVPR42600.2020.00975. URL <https://arxiv.org/abs/1911.05722>.
- [45] Stefan Hegselmann, Alejandro Buendia, Hunter Lang, Monica Agrawal, Xiaoyi Jiang, and David Sontag. TabLLM: Few-shot classification of tabular data with large language models. In *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2023. URL <https://arxiv.org/abs/2210.10723>.
- [46] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. In *International Conference on Learning Representations (ICLR)*, 2016. URL <https://arxiv.org/abs/1511.06939>.
- [47] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL <https://arxiv.org/abs/2203.15556>.
- [48] Noah Hollmann, Samuel Müller, Katharina Eggenberger, and Frank Hutter. TabPFN: A transformer that solves small tabular classification problems in a second. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=cp5PvcI6w8_.
- [49] Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeyer, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 637(8045):319–326, 2025. URL <https://www.nature.com/articles/s41586-024-08328-6>.
- [50] Yupeng Hou, Shanlei Mu, Wayne Xin Zhao, Yaliang Li, Bolin Ding, and Ji-Rong Wen. Towards universal sequence representation learning for recommender systems. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2022. URL <https://arxiv.org/abs/2206.05941>.
- [51] Yupeng Hou, Zhankui He, Julian McAuley, and Wayne Xin Zhao. Learning vector-quantized item representation for transferable sequential recommenders. In *Proceedings of the ACM Web Conference (WWW)*, 2023. URL <https://arxiv.org/abs/2210.12316>.
- [52] Yupeng Hou, Jiacheng Li, Zhankui He, An Yan, Xiusi Chen, and Julian McAuley. Bridging language and items for retrieval and recommendation. *arXiv preprint arXiv:2403.03952*, 2024.
- [53] Yupeng Hou, Junjie Zhang, Zihan Lin, Hongyu Lu, Ruobing Xie, Julian McAuley, and Wayne Xin Zhao. Large language models are zero-shot rankers for recommender systems. In *Advances in Information Retrieval – 46th European Conference on IR Research (ECIR)*, 2024. URL <https://arxiv.org/abs/2305.08845>.
- [54] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, Melbourne, Australia, July 2018. Association for Computational Linguistics. URL <https://aclanthology.org/P18-1031/>.
- [55] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022. URL <https://arxiv.org/abs/2106.09685>.

- [56] Xu Huang, Jianxun Lian, Yuxuan Lei, Jing Yao, Defu Lian, and Xing Xie. Recommender ai agent: Integrating large language models for interactive recommendations. *ACM Transactions on Information Systems*, 43, 2025. doi: 10.1145/3731446. URL <https://arxiv.org/abs/2308.16505>.
- [57] Yipo Huang, Xiangfei Sheng, Zhichao Yang, Quan Yuan, Zhichao Duan, Pengfei Chen, Leida Li, Weisi Lin, and Guangming Shi. Aesexpert: Towards multi-modality foundation model for image aesthetics perception. In *Proceedings of the 32nd ACM International Conference on Multimedia (MM)*, 2024. doi: 10.1145/3664647.3680649. URL <https://arxiv.org/abs/2404.09624>.
- [58] Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. On using very large target vocabulary for neural machine translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 1–10, 2015. doi: 10.3115/v1/P15-1001.
- [59] Yitong Ji, Aixin Sun, Jie Zhang, and Chenliang Li. A critical study on data leakage in recommender system offline evaluation. *ACM Transactions on Information Systems*, 41(3), 2023. doi: 10.1145/3569930.
- [60] Yangqin Jiang, Xubin Ren, Lianghao Xia, Da Luo, Kangyi Lin, and Chao Huang. RecGPT: A foundation model for sequential recommendation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2025. URL <https://doi.org/10.18653/v1/2025.emnlp-main.513>.
- [61] Daniel Kahneman. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, New York, 2011. ISBN 978-0374275631.
- [62] Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *IEEE International Conference on Data Mining (ICDM)*, pages 197–206, 2018. doi: 10.1109/ICDM.2018.00035. URL <https://arxiv.org/abs/1808.09781>.
- [63] Maxime Kawawa-Beaudan, Srijan Sood, Kassiani Papisotiriou, Daniel Borrajo, and Manuela Veloso. TradeFM: A generative foundation model for trade-flow and market microstructure. *arXiv preprint arXiv:2602.23784*, 2026. URL <https://arxiv.org/abs/2602.23784>.
- [64] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, 2017. URL <https://proceedings.neurips.cc/paper/2017/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html>.
- [65] Gautam Kedia and Stripe Machine Learning Team. Stripe’s payments foundation model. Post on X, May 7, 2025. <https://x.com/thegautam/status/1920198569308664169>, 2025.
- [66] Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*, 2019. URL <https://arxiv.org/abs/1909.05858>.
- [67] Ali Khanafer and Yang Liu. The generative recommender behind shopify’s commerce engine. Shopify Engineering, February 2026. URL <https://shopify.engineering/generative-recommendations>. Accessed 2026.
- [68] Myung Jun Kim, Léo Grinsztajn, and Gaël Varoquaux. CARTE: Pretraining and transfer for tabular learning. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024. URL <https://arxiv.org/abs/2402.16785>.
- [69] Walid Krichene and Steffen Rendle. On sampled metrics for item recommendation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1748–1757, 2020. doi: 10.1145/3394486.3403226.
- [70] Guillaume Lample, Alexis Conneau, Ludovic Denoyer, and Marc’Aurelio Ranzato. Unsupervised machine translation using monolingual corpora only. In *International Conference on Learning Representations (ICLR)*, 2018. URL <https://arxiv.org/abs/1711.00043>.
- [71] Doyup Lee, Chiheon Kim, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Autoregressive image generation using residual quantization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11513–11522, 2022. doi: 10.1109/CVPR52688.2022.01123. URL <https://arxiv.org/abs/2203.01941>.

- [72] Jiacheng Li, Ming Wang, Jin Li, Jinmiao Fu, Xin Shen, Jingbo Shang, and Julian McAuley. Text is all you need: Learning language representations for sequential recommendation. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2023. URL <https://arxiv.org/abs/2305.13731>.
- [73] Jinming Li, Wentao Zhang, Tian Wang, Guanglei Xiong, Alan Lu, and Gerard Medioni. Gpt4rec: A generative framework for personalized recommendation and user interests interpretation. *arXiv preprint arXiv:2304.03879*, 2023. URL <https://arxiv.org/abs/2304.03879>.
- [74] Darren Liu, Huayu Li, Raghav Boinepalli, Yuzhen Huang, Jiaqi Xu, et al. GEM training: How Meta doubled the efficiency of its LLM-scale ads foundation model. Engineering at Meta blog, August 3, 2026. <https://engineering.fb.com/2026/08/03/ml-applications/training-gem-at-llm-scale-meta-ads-recommendation-foundation-model/>, 2026.
- [75] Junling Liu, Chao Liu, Renjie Lv, Kang Zhou, and Yan Zhang. Is chatgpt a good recommender? a preliminary study. *arXiv preprint arXiv:2304.10149*, 2023. URL <https://arxiv.org/abs/2304.10149>.
- [76] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H. Chi. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018. doi: 10.1145/3219819.3220007.
- [77] Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. The natural language decathlon: Multitask learning as question answering. *arXiv preprint arXiv:1806.08730*, 2018. URL <https://arxiv.org/abs/1806.08730>.
- [78] Meta. Meta’s generative ads model (GEM): The central brain accelerating ads recommendation AI innovation. Engineering at Meta blog, November 10, 2025. <https://engineering.fb.com/2025/11/10/ml-applications/metas-generative-ads-model-gem-the-central-brain-accelerating-ads-recommendation-ai-innovation/>, 2025.
- [79] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper_files/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.
- [80] Maxim Milakov and Natalia Gimelshein. Online normalizer calculation for softmax. *arXiv preprint arXiv:1805.02867*, 2018. URL <https://arxiv.org/abs/1805.02867>.
- [81] Milton Llera Montero, Casimir JH Ludwig, Rui Ponte Costa, Gaurav Malhotra, and Jeffrey Bowers. The role of disentanglement in generalisation. In *International Conference on Learning Representations*, 2021. URL <https://openreview.net/forum?id=qBH974jKUVy>.
- [82] Netflix Technology Blog. Foundation model for personalized recommendation. <https://netflixtechblog.com/foundation-model-for-personalized-recommendation-1a0bd8e02d39>, March 2025.
- [83] Maxim Ostroukhov, Ruslan Mikhailov, Vladimir Iashin, Artem Sokolov, Andrei Akshonov, Vitaly Protasov, Andrey Goncharov, Dmitrii Beloborodov, Vince Mullin, Roman Y. Enzmann, Georgios Kolovos, Jason Renders, Pavel Nesterov, and Anton Repushko. PRAGMA: Revolut foundation model. *arXiv preprint arXiv:2604.08649*, 2026. URL <https://arxiv.org/abs/2604.08649>.
- [84] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL <https://arxiv.org/abs/2203.02155>.
- [85] Jared Palmer. Kev: Small Jev-like decision models you can train and run yourself. <https://github.com/jaredpalmer/kev>, 2026.
- [86] Victor Hugo Panisa. Von: An Open-Source System One Decision Model. <https://github.com/wfzyx/von>, 2026.

- [87] Ofir Press and Lior Wolf. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 157–163, 2017. doi: 10.18653/v1/E17-2025. URL <https://arxiv.org/abs/1608.05859>.
- [88] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. OpenAI technical report, 2018. URL https://cdn.openai.com/research-cover-s/language-unsupervised/language_understanding_paper.pdf.
- [89] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. OpenAI technical report, 2019. URL https://cdn.openai.com/better-1-language-models/language_models_are_unsupervised_multitask_learners.pdf.
- [90] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning (ICML)*, 2021. URL <https://arxiv.org/abs/2103.00020>.
- [91] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL <https://arxiv.org/abs/1910.10683>.
- [92] Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan H. Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Q. Tran, Jonah Samost, Maciej Kula, Ed H. Chi, and Maheswaran Sathiamoorthy. Recommender systems with generative retrieval. In *Advances in Neural Information Processing Systems*, 2023. URL <https://arxiv.org/abs/2305.05065>.
- [93] Xiang-Rong Sheng, Liqin Zhao, Guorui Zhou, Xinyao Ding, Binding Dai, Qiang Luo, Siran Yang, Jingshan Lv, Chi Zhang, Hongbo Deng, and Xiaoqiang Zhu. One model to serve all: Star topology adaptive recommender for multi-domain CTR prediction. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management (CIKM)*, 2021. doi: 10.1145/3459637.3481941.
- [94] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024. URL <https://arxiv.org/abs/2408.03314>.
- [95] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014. ISSN 1532-4435. URL <https://jmlr.org/papers/v15/srivastava14a.html>.
- [96] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformers. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 1441–1450. ACM, 2019. doi: 10.1145/3357384.3357895.
- [97] Hongyan Tang, Junling Liu, Ming Zhao, and Xudong Gong. Progressive layered extraction (PLE): A novel multi-task learning (MTL) model for personalized recommendations. In *Proceedings of the 14th ACM Conference on Recommender Systems (RecSys)*, 2020. doi: 10.1145/3383313.3412236.
- [98] Jiakai Tang, Sunhao Dai, Teng Shi, Jun Xu, Xu Chen, Wen Chen, Jian Wu, and Yuning Jiang. Think before recommend: Unleashing the latent reasoning power for sequential recommendation. *arXiv preprint arXiv:2503.22675*, 2025. URL <https://arxiv.org/abs/2503.22675>.
- [99] TRI LBM Team, Jose Barreiros, Andrew Beaulieu, Aditya Bhat, Rick Cory, Eric Cousineau, Hongkai Dai, Ching-Hsin Fang, Kunimatsu Hashimoto, Muhammad Zubair Irshad, Masha Itkina, Naveen Kuppaswamy, Kuan-Hui Lee, Katherine Liu, Dale McConachie, Ian McMahon, Haruki Nishimura, Calder Phillips-Grafflin, Charles Richter, Paarth Shah, Krishnan Srinivasan, Blake Wulfe, Chen Xu, Mengchao Zhang, Alex Alspach, Maya Angeles, Kushal Arora, Vitor Campagnolo Guizilini, Alejandro Castro, Dian Chen, Ting-Sheng Chu, Sam Creasey, Sean Curtis, Richard Denitto, Emma Dixon, Eric Dusel, Matthew Ferreira, Aimee Goncalves, Grant Gould, Damrong Guoy, Swati Gupta, Xuchen Han, Kyle Hatch, Brendan Hathaway, Allison Henry, Hillel Hochsztein, Phoebe Horgan, Shun Iwase, Donovan Jackson, Siddharth Karamcheti, Sedrick Keh, Joseph Masterjohn, Jean Mercat, Patrick Miller, Paul Mitiguy, Tony Nguyen, Jeremy Nimmer, Yuki Noguchi, Reko Ong, Aykut Onol, Owen Pfannenstiehl, Richard Poyner, Leticia Priebe Mendes Rocha, Gordon Richardson, Christopher Rodriguez, Derick Seale, Michael Sherman, Mariah Smith-Jones, David Tago, Pavel Tokmakov,

- Matthew Tran, Basile Van Hoorick, Igor Vasiljevic, Sergey Zakharov, Mark Zolotas, Rares Ambrus, Kerri Fetzer-Borelli, Benjamin Burchfiel, Hadas Kress-Gazit, Siyuan Feng, Stacie Ford, and Russ Tedrake. A careful examination of large behavior models for multitask dexterous manipulation. *arXiv preprint arXiv:2507.05331*, 2025. URL <https://arxiv.org/abs/2507.05331>.
- [100] Yancheng Wang, Ziyang Jiang, Zheng Chen, Fan Yang, Yingxue Zhou, Eunah Cho, Xing Fan, Yanbin Lu, Xiaojiang Huang, and Yingzhen Yang. RecMind: Large language model powered agent for recommendation. In Kevin Duh, Helena Gomez, and Steven Bethard, editors, *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 4351–4364, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-naacl.271. URL <https://aclanthology.org/2024.findings-naacl.271/>.
- [101] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems, NIPS ’22*, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088. URL <https://arxiv.org/abs/2201.11903>.
- [102] Erik Wijmans, Brody Huval, Alexander Hertzberg, Vladlen Koltun, and Philipp Krähenbühl. Cut your losses in large-vocabulary language models. In *International Conference on Learning Representations (ICLR)*, 2025. URL <https://arxiv.org/abs/2411.09009>.
- [103] Gerald Woo, Chenghao Liu, Akshat Kumar, Caiming Xiong, Silvio Savarese, and Doyen Sahoo. Unified training of universal time series forecasting transformers. In *International Conference on Machine Learning (ICML)*, 2024. URL <https://arxiv.org/abs/2402.02592>.
- [104] Max Wornow, Yikuan Xu, Rishav Thapa, et al. The shaky foundations of large language models and foundation models for electronic health records. *npj Digital Medicine*, 6:135, 2023. doi: 10.1038/s41746-023-00879-8. URL <https://doi.org/10.1038/s41746-023-00879-8>.
- [105] Junda Wu, Cheng-Chun Chang, Tong Yu, Zhankui He, Jianing Wang, Yupeng Hou, and Julian McAuley. Coral: Collaborative retrieval-augmented large language models improve long-tail recommendation. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2024. doi: 10.1145/3637528.3671901. URL <https://arxiv.org/abs/2403.06447>.
- [106] Likang Wu, Zhi Zheng, Zhaopeng Qiu, Hao Wang, Hongchao Gu, Tingjia Shen, Chuan Qin, Chen Zhu, Hengshu Zhu, Qi Liu, Hui Xiong, and Enhong Chen. A survey on large language models for recommendation. *World Wide Web*, 27, 2024. doi: 10.1007/s11280-024-01291-2. URL <https://arxiv.org/abs/2305.19860>.
- [107] Qiuling Xu, Ko-Jen Hsiao, and Moumita Bhattacharya. Towards generalizable and efficient large-scale generative recommenders. In *Proceedings of the 20th ACM Conference on Recommender Systems (RecSys)*, 2026. doi: 10.1145/3773078.3831905. URL <https://arxiv.org/abs/2605.23312>.
- [108] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. URL <https://arxiv.org/abs/1906.08237>.
- [109] Chin-Chia Michael Yeh, Uday Singh Saini, Xin Dai, Xiran Fan, Shubham Jain, et al. TREASURE: The Visa payment foundation model for high-volume transaction understanding. *arXiv preprint arXiv:2511.19693*, 2025. URL <https://arxiv.org/abs/2511.19693>.
- [110] Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee Kumthekar, Zhe Zhao, Li Wei, and Ed Chi. Sampling-bias-corrected neural modeling for large corpus item recommendations. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 269–277, 2019. doi: 10.1145/3298689.3346996.
- [111] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Albeti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. URL <https://arxiv.org/abs/2007.14062>.
- [112] Jiaqi Zhai, Lucy Liao, Xing Liu, Yueming Wang, Rui Li, Xuan Cao, Leon Gao, Zhaojie Gong, Fangda Gu, Jiayuan He, Yinghai Lu, and Yu Shi. Actions speak louder than words: Trillion-parameter sequential transducers for generative recommendations. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on*

Machine Learning, volume 235 of *Proceedings of Machine Learning Research*, pages 58484–58509. PMLR, 21–27 Jul 2024. URL <https://proceedings.mlr.press/v235/zhai24a.html>.

- [113] Buyun Zhang, Liang Luo, Yuxin Chen, Jade Nie, Xi Liu, and Others. Wukong: Towards a scaling law for large-scale recommendation. In *International Conference on Machine Learning (ICML)*, 2024. URL <https://arxiv.org/abs/2403.02545>.
- [114] Ruijie Zheng, Yongyuan Liang, Shuaiyi Huang, Jianfeng Gao, Hal Daumé III, Andrey Kolobov, Furong Huang, and Jianwei Yang. Tracevla: Visual trace prompting enhances spatial-temporal awareness for generalist robotic policies. In *International Conference on Learning Representations (ICLR)*, 2025. URL <https://arxiv.org/abs/2412.10345>.
- [115] Guorui Zhou, Honghui Bao, Jiaming Huang, Jiaxin Deng, Jinghao Zhang, Qiang Luo, Ruiming Tang, et al. Openonerec technical report: An open foundation model and benchmark to accelerate generative recommendation. *arXiv preprint arXiv:2512.24762*, 2025. URL <https://arxiv.org/abs/2512.24762>. OneRec Team, Kuaishou.
- [116] Zikang Zhou, Haibo Hu, Xinhong Chen, Jianping Wang, Nan Guan, Kui Wu, Yung-Hui Li, Yu-Kai Huang, and Chun Jason Xue. Behaviorgpt: Smart agent simulation for autonomous driving with next-patch prediction. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL <https://arxiv.org/abs/2405.17372>.
- [117] Bingzhao Zhu, Xingjian Shi, Nick Erickson, Mu Li, George Karypis, and Mahsa Shoaran. XTab: Cross-table pretraining for tabular transformers. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023. URL <https://arxiv.org/abs/2305.06090>.
- [118] Feng Zhu, Yan Wang, Chaochao Chen, Jun Zhou, Longfei Li, and Guanfeng Liu. Cross-domain recommendation: Challenges, progress, and prospects. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI)*, 2021. doi: 10.24963/ijcai.2021/639.

A Extended Definition and Positioning

This appendix expands the definition given in Section 1 and the reasoning behind it. The full position is developed in a companion paper [33]; what follows is the part most directly relevant to the design choices in this work.

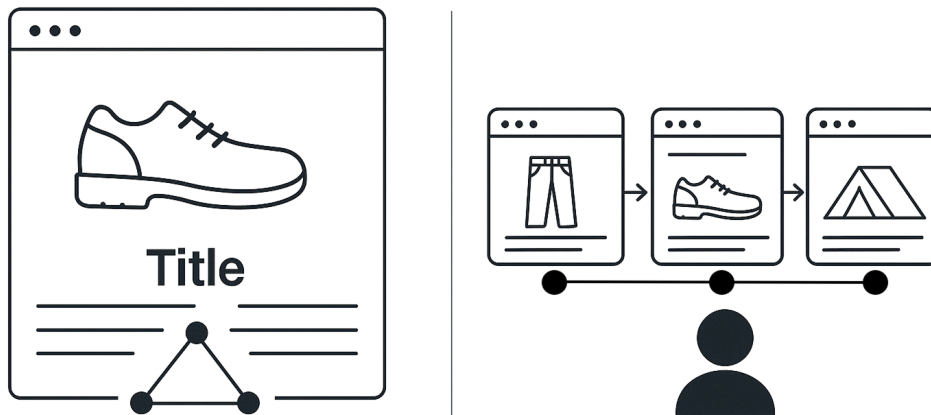


Figure 15: Contrasting data perspectives: LLMs and VLMs derive meaning from static co-occurrences (e.g., on webpages) (left), while LBM derive meaning from chronological behavioral sequences (right).

A.1 The Comparison Axes

Parameters. The table reports **non-embedding** parameters, the only ones comparable across these architectures, since a recommender over a billion-item vocabulary can hold a trillion parameters in lookup tables while computing with very few. That is why HSTU’s headline 1.5T is not the relevant quantity, and why TREASURE, three layers at width 256, sits some two orders of magnitude below the 0.4–1.8B total the paper reports. Dashes mean not reported, so the figures given are the largest *disclosed* rather than the largest.

How each count was obtained. Few industrial behavioral models in the public record report a comparable figure: Shopify gives 8B total in a talk, split between embedding tables and the encoder in unstated proportion, so no non-embedding count can be estimated for it; the largest estimable is PRAGMA at 1B total parameters, of which we estimate roughly 0.9B is compute-bearing, and the largest reported is Netflix’s 1B backbone, counted without its embedding and decoding layers [107], which leaves better than a tenfold gap to ours. HSTU’s headline is far larger—1.5 trillion parameters—but that number sums embedding and non-embedding parameters, a distinction the paper draws explicitly and then does not quantify; its largest reported configuration implies on the order of 0.13B compute-bearing parameters. The remaining industrial figures come from the same procedure applied to published configurations—PRAGMA’s 72 encoder layers of a reported 1B total, nuFormer’s 24 layers of 330M, TradeFM’s 32 grouped-query layers of 524M, TREASURE’s three layers at width 256—and only TransactionGPT and Netflix report the quantity directly. For all but the last the estimate is close to the reported total, because the vocabularies are small, some 28k value tokens for PRAGMA and 16k for TradeFM, so the ambiguity that makes HSTU’s headline unusable barely bites elsewhere. The gap is largest for TREASURE: those three layers amount to roughly 5M compute-bearing parameters, while the models its scaling study reports total 0.4–1.8B, nearly all of it in embedding tables. By its headline parameter count, a model can therefore be two orders of magnitude larger than the model that runs—the clearest case for reporting the non-embedding count, and the reason we report it. The two entries in the table nearest ours in size sit outside this block entirely: OneRec-Foundation’s 7B and TabuLa-8B’s 7B are the language models they are built on, Qwen3-8B and Llama-3-8B without their embeddings, a size inherited from the backbone rather than earned on behavioral data, which is why the comparison in Section 6.4 runs over behavioral foundation models. Our corpus is likewise the largest disclosed outside our own earlier work [36], though by a narrower margin than the parameter count: the largest reported event count is HSTU’s 100 billion examples against our 150B actions, and the largest reported token count is Netflix’s 2T per training cycle against our 3.6T, under a factor of two. The one figure that may be comparable is Shopify’s “trillion-level” tokens per epoch, an order of magnitude given in a talk rather than a number.

The academic baselines sit at the opposite extreme. None of them reports a parameter count either, so we counted from released code and checkpoints: the trained non-embedding parameters run from under a million in ZESRec and XTab, through three million in UniSRec and VQ-Rec and twenty-one million in RecGPT, to 106M in RecFormer, the only one of them that fine-tunes a full language backbone—four to five orders of magnitude below ours, on corpora of a few

million sequences. TabPFN is smaller still, and version 2 is smaller than version 1, at 7.2M against 25.8M. Where a row of the table names two models we report the larger and give the other here: XTab’s transferable backbone is 0.8M against CARTE’s 9.3M, and TabLLM does not pretrain on tables at all, training some 1M (IA)³ adapter parameters per task on an off-the-shelf 11.1B T0pp. These counts change how their transfer results should be read: three of the five transferable recommenders are a few-million-parameter head on a *frozen* BERT whose item vectors are precomputed offline, so when such a model generalizes to an unseen catalog, the evidence is in the first instance about what BERT knows about product descriptions, and only derivatively about behavioral pretraining. Separating those two explanations is what the comparison in this paper is built to do. Both statements are about the public record rather than about what exists: these models are trained on continuous internal streams whose size is never published, and Meta’s more recent ads foundation model, GEM, is described only as having billions of dense parameters alongside trillions of sparse ones [78, 74]. Size matters here for what it lets us test: transfer at 10^4 – 10^5 times the pretraining scale of prior transfer studies, and transfer across industries, which to our knowledge has not been tested at any scale.

The marks against individual cells. *Multiple tasks, but all internal to the operating company and not externally reproducible. [¶]Rating prediction, recommendation, explanation and summarization have all been posed to prompted LLMs, but the LLM either ranks a bounded candidate list supplied in context or generates text that a separate retriever matches to the catalog; it never scores the full catalog itself. [#]Transfers to ten held-out Amazon datasets, but by fine-tuning on the target data under leave-one-out splits, not zero-shot. [°]Our count, excluding embedding tables, from the published configuration or the released weights; only Netflix, TransactionGPT, TabPFN v1 and CARTE report this quantity directly, and Appendix A.1 gives the accounting for every estimate. For the industrial models the estimate is close to the reported total because their vocabularies are small—TREASURE excepted, its merchant table dwarfing the three layers that compute; HSTU’s cell is the largest configuration it reports testing, its headline 1.5T summing both kinds of parameter under a distinction it draws but does not quantify. [†]Trained parameters only: a small trained module on a *frozen* text encoder (BERT-base, or MPNet in RecGPT) whose item vectors are precomputed offline, adding 85M of inference compute apiece that is not learned from behavior. Counting it leaves the whole block at or below 0.11B. ZESRec’s cell is an upper bound, its trainable width never having been published. ^hRow names two models: we report the larger, with the other in Appendix A.1. TabPFN’s corpus is version 2’s; version 1 trained on 9.2M synthetic datasets. ⁱAs reported: backbone parameters excluding embedding and decoding layers, and behavior tokens per training cycle [107]. ^sShopify publishes no paper: the 8B is a *total* including embedding tables, split from the encoder in unstated proportion, so its cell is an upper bound on the column’s quantity, and the corpus “trillion-level tokens per epoch,” both stated in a talk [3], with 100B parameters stated as the roadmap; the written account [67] gives no figures. [¶]Label prediction over a fixed-width table of i.i.d. rows; applicable to any domain, but unable to rank even moderately sized catalogs, read images, or represent user sequences well, so most tasks here cannot be posed to them at all. [†]Sequence-length extrapolation on the same dataset, not transfer to a new one. [°]Cold-starts unseen items *within its own catalog*, by an attention over title age that defers to IDs once interaction data exists—not evaluation on a held-out dataset. ^mSeveral countries, but with a separately pretrained model per market. ^uInferred from first-party descriptions; Stripe publishes no architecture-level account, so this cell is less firmly sourced. ^gThe one industrial exception: US equities transferred zero-shot to Japanese and Chinese markets. Its inputs carry no instrument identifier, and it is transfer across venues within one asset class rather than to a new catalog or task. It is also the only row in this block that sequences an *instrument* rather than a person or account, so it is behavioral in the aggregate sense—many agents against one series—rather than in the per-agent sense the rest of the block and this paper use (Section 6.4). [‡]Against baselines trained on 10–50% of the target data. ^zFine-tuned on the target in its own paper; evaluated zero-shot by others [72]. ⁱNo weights are updated, but labeled rows of the target table are given as context. [§]Against fully trained specialists under full-catalog ranking with temporal splits.

Carrier. The representation through which an event enters the model, and hence the only thing that can carry it to an unseen catalog. The platform’s own *IDs* cannot leave it, a new catalog having no rows in the embedding table. Typed *schema* or *metadata* fields travel only where a schema is shared, and their high-cardinality members are IDs in disguise. *Text* travels wherever language does. A synthetic *prior* travels anywhere. *Multimodal* event features—text, imagery and numeric fields jointly—are ours. Hybrids list the dominant component first.

Geometry. What makes two events close in the resulting space: behavioral co-occurrence, linguistic similarity of descriptions, or a prior over causal graphs. Read against the carrier column, it gives the table’s central contrast: every industrial model has a behavioral geometry but a carrier that cannot leave its platform, while the models that do travel take their geometry from language.

A.2 Where Meaning Comes From

Large Behavioral Models (LBMs) sit within the broader class of foundation models: any model trained on broad data (typically via self-supervision at scale) that can be adapted to a wide range of downstream tasks [9]. This foundation is shared with Large Language Models (LLMs) and Vision-Language Models (VLMs), but LBMs diverge in how data is interpreted and meaning derived.

LLMs and VLMs primarily learn from statistical co-occurrences of text and images in contexts like webpages (Figure 15, left). The meaning of pixels emerges from their co-occurrence with other pixels and surrounding words, while words derive meaning from adjacent text and images. Webpages, however, are manufactured artifacts, not chronological behaviors—they are outcomes of actions, and are often acted upon themselves (Figure 15, right).

The behavioral perspective is instead dynamic and sequence-oriented, and encompasses two views: (i) the chronological actions of the creator producing the content, which define elements like pixels and words through their co-occurrence in the creation sequence; and (ii) the chronological actions of users interacting with the content, where meaning arises from co-occurrences in consumption or engagement sequences. Any text, image, or video rarely represents a chronological sequence of individual human actions; it is the aggregated outcome of those actions across individuals—consider the iterative process of writing, editing, and refining this very paper, interspersed with pauses and external influences. The underlying actions are a different and less curated perspective on the same object—recording what people did rather than what they chose to share. The data is also abundant: estimates we conducted with major data providers suggest behavioral data in retail and payments alone may exceed internet text by a factor of 100–1000 [33].

Behavior is therefore a complementary, often orthogonal source of meaning to the text and images LLMs and VLMs learn from. Robotics illustrates this [114]: models trained solely on static images or text may misinterpret scenes for lack of action context, while training on behavioral trajectories improves interpretation, because elements are understood through their roles in action flows rather than isolated co-occurrences. Even averaging all behaviors involving an image typically does not replicate the meanings that VLMs and LLMs capture from pixel occurrences and text–image pairs, and vice versa. The classic “invisible gorilla” experiment makes the same point about perception: participants counting basketball passes often fail to notice a gorilla walking through the scene, because behavioral context filters information rather than merely adding to it. LLMs derive meaning from linguistic co-occurrences, VLMs from visual and textual co-occurrences, and LBMs from behavioral co-occurrences; we believe a complete machine understanding of people requires all three.

Over time, behavioral models for different domains and time scales may converge into fewer, more general LBMs; the transfer results in Section 3 are an early indication of how far a single set of behavioral weights can reach.

A.3 Complementarity and Compounding

LBMs complement LLMs rather than compete with them. LLMs reason about behavior in language space and must reach the action space through slow external tools; LBMs are trained directly in the action space and can search, rank, and simulate trajectories natively, at a fraction of the cost and latency (Table 3). Conversely, LLMs supply linguistic interface and world knowledge that LBMs lack. The most successful deployments we have built are two-sided: the LBM serves as a behavioral-intelligence tool for the LLM, and the LLM as a language interface for the LBM (Section 6).

Deployed foundation models compound. Better models generate better behavioral data, which trains better models. Across several years of production deployments we estimate this “interest rate” of foundational AI at roughly 2.5% average annual conversion uplift from retraining and redeployment alone, excluding gains from newly enabled products [33]. This compounding is why our evaluation emphasizes deployment-grade metrics alongside offline benchmarks: serving cost in Table 3 and A/B outcomes in Table 4.

A.4 Systems 1, 2, and 0

Over the course of this six-year project, a natural structure has emerged in our approach to developing foundation models for behavior. We have found it useful to categorize our efforts into three systems, inspired by Kahneman [61], with an additional System 0.

System 1 encompasses fast, black-box applications of models for consumer use, such as search, recommendations, fraud detection, and dynamic pricing.

System 2 involves analyzing model representations to yield interpretable results, often for business intelligence, such as market analysis, operational expansions or contractions, and assessments of assortment strengths and weaknesses.

System 0 focuses on data collection and engineering, including understanding model inputs.

This paper primarily addresses System 1, while also discussing the others. The three feed one another: we examine the data, train and evaluate models, and study their representations, which informs better data collection and in turn better models. Expertise in one therefore requires proficiency in the others. Foundation models also change what data an enterprise should collect, a shift enterprises often overlook. The explainability features that earn organizational trust shape model training and design, and the tasks that most affect end users point business intelligence to the analyses with the most impact.

B Architecture and Objectives

This appendix and the two that follow collect the techniques, design discussions, and engineering details behind the recipe summarized in Section 2: this one covers the model and its objectives, Appendix C the data, and Appendix D training, systems, and metrics. These appendices record techniques and experiments from several BehaviorGPT generations, not only the BehaviorGPT-v4 recipe.

B.1 Output Space and Candidate Representations

Output versus Input Space: Digesting and predicting actions are distinct tasks in behavioral modeling [87]. This distinction is sharper in LBM than in LLMs. Embedding a sequence of products scales with the batch size and sequence length ($B \times L$). However, predicting the next product requires retrieving from a candidate set, scaling as $B \times L \times C$, where C , the size of the candidate set, can exceed 10 billion for all possible actions. Thus, embedding the candidate space is up to C times more computationally expensive than embedding the input space. We explore several approaches to address this challenge:

- *One-Hot Encodings for Sources and Targets:* This approach scales as $B \times L \times C$, with each operation being computationally inexpensive due to the simplicity of one-hot embeddings. However, one-hot encodings do not naturally generalize to new products, assortments, or tasks (e.g., search), unlike feature-based representations.
- *Feature Embedder for Sources and Targets:* This method also scales as $B \times L \times C$, but each operation is costly, as it involves learning to map text and image features to vectors. The extra cost buys richer, more generalizable representations.
- *Feature Embedder for Sources and Text-Generation for Targets:* Instead of retrieving the next action from the candidate set, this approach generates the text description of the next product as a bag-of-words, reducing the target space to the number of tokens (e.g., 100,000) rather than the full candidate set (beyond 10 billion). While this resembles quantization, it does not inherently provide a fast retrieval mechanism, as describing a product does not directly enable selecting it from a large assortment during inference. However, this method yields strong embedders that can be frozen in the second training stage for retrieval tasks. Generating text descriptions can also support downstream tasks, such as autocomplete or personalized search refinement, though static descriptions risk memorization, necessitating augmentation techniques [32].
- *Feature Embedder for Sources and Latent Vectors as Targets:* Using pre-trained image or text embedders, this approach predicts the latent vector of the next action via an L2 loss, scaling as $B \times L$ and requiring less computation. At inference, retrieval involves finding the closest product in L2 distance, assuming a high-quality pre-trained embedder is available.
- *Predicting Relative Positions:* To reduce the target space, this method predicts the relative positions of actions within a sequence rather than their identities or features. By masking actions and training the model to infer their positions, it learns to understand action relationships [18]. The candidate space is reduced to the number of sequence positions, which is orders of magnitude smaller than the number of possible actions.
- *Mixing Multiple Objectives:* Combining multiple objectives, such as retrieving products, predicting their text tokens, and minimizing L2 loss to semantic vectors, improves pretraining performance; the challenge is to balance that gain against simplicity.
- *Small and Large Embedders:* Embedders capture both global information (e.g., popularity, general category) and local information (e.g., specific brand, color, or price preferences). Using multiple embedders tailored to these different scales of information can improve performance. This is further explored in the “Global and Local Loss” section below.

Time Since and Popularity Prior work has recognized that the age of an action or item, defined as the time since its introduction, is a strong predictor of its likelihood of occurrence [23]. However, incorporating this feature into

Transformer-based autoregressive models requires careful engineering to ensure scalability. For a batch size B , sequence length S , and candidate set size C_B , the output logits have dimensions $B \times S \times C_B$. We aim to incorporate two features for each logit: (i) the time since an item’s introduction, computed as the difference between the static introduction time of an item in C_B and the event time in $B \times S$, and (ii) the cumulative interactions with the item (sales, in retail) from its introduction to the current user event.

For (i), the time-since-introduction can be cached with a memory footprint of order $(B \times S) + C_B$ across all training data, enabling computation of $B \times S \times C_B$ values at training time. For (ii), caching sales data for each target-user-event triplet would require a prohibitive $B \times S \times C_B$ cache. Instead, we propose storing sales data as a segmented time series with K segments (e.g., weeks or months), reducing the cache size to $C_B \times K$. Alternatively, fitting a simple function to sales over time allows querying continuous time values, further compressing the representation. This results in a $C_B \times K$ matrix, indexed by the $B \times S$ event times to produce $B \times S \times C_B$ values.

Given hidden embeddings $h \in \mathbb{R}^{B \times S \times d}$ and target embeddings $c \in \mathbb{R}^{C_B \times d}$, computing $B \times S \times C_B$ embeddings to incorporate time-since-introduction and sales-since-introduction is computationally expensive due to the high dimensionality of d . Instead, we incorporate these features at the logit level, where the output of the matrix product $hc^T \in \mathbb{R}^{B \times S \times C_B}$ is one-dimensional per element. Using small neural networks f_1 and f_2 , we compute:

$$f_2(hc^T + f_1(t_1, t_2)),$$

where $t_1, t_2 \in \mathbb{R}^{B \times S \times C_B}$ represent time-since-introduction and sales-since-introduction, respectively. This approach is computationally feasible and can be introduced during finetuning after initial training without these features. At inference, synchronized user event times further improve efficiency.

Incorporating temporal interaction trends across users enhances model performance and subsumes the need for separate time-since-introduction and popularity features, as these are implicitly encoded in the interaction trends. For each user, sequence, and target action, we include a historical interaction vector (e.g., a 36-dimensional vector capturing monthly interactions over the past 36 months), providing richer context for modeling action popularity and temporal dynamics. This results in $t_2 \in \mathbb{R}^{B \times S \times C_B \times 36}$, which is processed by a linear layer or a small neural network to produce $\mathbb{R}^{B \times S \times C_B}$ for incorporation into the logits.

As implemented. In BehaviorGPT-v4 both features enter as additive logit residuals that are bilinear in the hidden state. A candidate’s age at the event is bucketed by month into 13 buckets, and its sales up to that age are binned by $\log_{10}$ into 12 levels. Each bucket has a learned vector whose inner product with h is added to the logit, giving the form of Appendix D.4. Candidates introduced after the event’s timestamp are masked out of the softmax (the target never is), so no position is scored against an item that did not yet exist. Because the residuals depend only on the query and the bucket, serving folds them into the cached candidate table at the request’s timestamp.

Cache (MoCo) for Candidates Inspired by Momentum Contrast (MoCo) [44], we have experimented with a cache of candidates from the two preceding batches during Stage 1 training. It slightly improved the performance of the embedder by providing a more diverse and temporally consistent set of candidates for contrastive learning.

B.2 Learning to Tokenize (Quantization)

Quantization techniques such as Vector Quantized Variational Autoencoders (VQ-VAE) and Residual Quantized Variational Autoencoders (RQ-VAE) compress behavioral sequence data effectively. Generative recommendation on top of them, decoding with beam search, is expensive and can lower accuracy. As with the feature embedders, the quantization should be learned end-to-end with the sequential autoregressive task so that it serves the predictive objective. We find quantization helps most in the input space, where it compresses sequences while preserving their behavioral patterns. We also adopt multi-target prediction, following its use in large language models [42], to make the quantization more stable and efficient.

End-to-end quantization. Two points are critical. First, it is not sufficient to learn the quantization by sampling from a fixed set of actions or a dataset using auto-encoders — this will learn to semantically embed items by co-occurrence within the features and the set, rather than letting meaning be defined by co-occurrence in behavior. Thus, quantization should be trained end-to-end on the sequential autoregressive pre-training task of interest. Second, we find the quantization of sources and candidates (in target space) are very different, and thus one cannot simply quantize one of them and use it for both. See Figure 16 for how this can be done end-to-end. We quantize the source embedding with a codebook, then also quantize every candidate, and apply a regular softmax in this space. The disadvantage is that this forgoes the benefits of quantization, though it can serve as a first step, with the learned quantization then used for compression and generative recommendation.

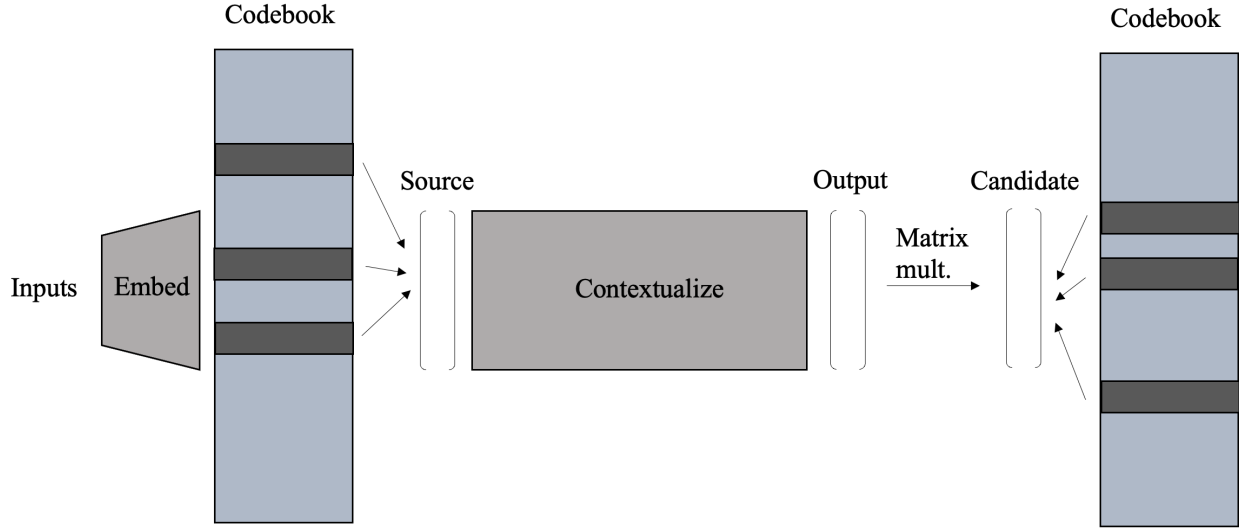


Figure 16: End-to-end quantization integrated into the autoregressive pre-training task. Unlike typical pre-hoc approaches that quantize data using a fixed codebook before training, this method jointly learns the quantization codebook with the model. Both source and candidate embeddings are quantized using this shared codebook during training, allowing the learned representations to reflect behavioral co-occurrence. This setup does not yet use quantization for compression or retrieval, but the learned codebook can serve those uses later.

B.3 Reversing Stage 1 and Stage 2 (Unsupervised Translation)

The two-stage process can in principle be reversed: first train a robust contextualizer, then fine-tune the embedder against it frozen. Consider a scenario with abundant one-hot encoded events corresponding to products. Using one-hot embeddings, a large-scale contextualizer can be trained to predict the next event based on prior events. As noted earlier, this approach struggles to generalize to new products due to the limitations of one-hot encodings. However, suppose a smaller dataset with rich features, such as product descriptions and images, is available. We can fine-tune the model by introducing a feature embedder and training it on this smaller dataset, replacing the one-hot embedder. Optionally, freezing the contextualizer during this phase constrains the embedder to produce representations that align with the contextualizer’s learned predictive task.

This approach differs from an alternative that assumes products overlap between datasets and uses them to map product descriptions and images to their corresponding one-hot embeddings. Unlike the mapping approach, reversing the stages does not require explicit correspondences between one-hot encoded events and feature-rich data. Instead, it relies solely on congruous sequences—sequences that share similar behavioral patterns—so the embedder learns representations compatible with the contextualizer’s objective.

B.4 Time-Delay Events

We make each time delay between events its own event, which the model both predicts and conditions on. At evaluation, specifying a short delay before the next product yields more relevant, less diverse results, and a longer delay more far-fetched, more diverse ones. Time delays could be embedded using a linear layer to better embed continuous variables such as time and be predicted via regression (or grouped together to discrete targets and input).

We experimented with discrete versus continuous encodings of times, and found that discrete input and target worked the best. Optimal was 30 buckets of time, with an equal number of items in each bucket, based on training data. However, we also found that for most tasks, predicting time delay (and not just having it as input) is often not an optimal use of compute. We have also experimented with introducing time delay later in training, which in those experiments gave better results.

B.5 Losses and Objectives

For LLMs, a single auto-regressive cross-entropy loss is optimized, mainly because the candidate space (the vocabulary) is fixed and small enough to compute a complete distribution over at every step. For behavior, this is different. We

typically need and want to sample the candidate space, but how we sample it affects what the loss actually is and what the model ends up learning. With hard-negative sampling, for example, the model learns to discern between similar items (what we call *local* information) but may suffer in its ability to understand the overall popularity of an event, item, or product (*global* information)—e.g., the difference between a popular and an unpopular Nike shoe. We address this by having two losses, one for local information and one for global information.

Imagine using the candidates that are in the current batch (from training sequences) as the samples for the loss. With a batch of 32 sequences, the candidates from a position’s own sequence tend to be similar to its target, so a fraction of the batch candidates act as hard negatives. If we instead sample from the full space of actions, say 10B, most of them will be far away from the true targets, in comparison. What the model learns from a sampled loss depends on how the candidates are sampled, which we now make explicit. For a position with history s , the target is $y \sim p(\cdot | s)$, the candidate set is $C = \{y, x_1, \dots, x_K\}$ with negatives x_k drawn i.i.d. from a proposal q , and the model scores a candidate as $f(x, s) = x^\top h(s)$, where $h(s)$ is the contextualizer’s output. The loss is

$$\ell = -\log \frac{e^{f(y, s)}}{\sum_{x \in C} e^{f(x, s)}}. \quad (5)$$

Batch candidates are the targets of other positions, so their proposal is the marginal frequency of events, $q(x) \propto p(x)$.

Fixed h . Suppose first that h is one vector shared by every position. The target distribution it faces is then the marginal $p(x)$, and an item’s chance of appearing as the target is proportional to its chance of appearing as a negative. The optimum, derived below with $p(x | s)$ replaced by $p(x)$, is $f(x) = \log p(x) - \log q(x) + c = c$: every candidate receives the same score, so a fixed h trained on batch candidates learns nothing about popularity.

History-dependent h . The trained model has $h = h(s)$, which differs per position, so the fixed case does not settle what it learns. As K grows, $\frac{1}{K} \sum_k e^{f(x_k, s)} \rightarrow \mathbb{E}_{x \sim q} e^{f(x, s)}$, the target’s own term in the denominator becomes negligible, and up to the constant $\log K$ the expected loss at history s is

$$\mathcal{L}(s) = -\mathbb{E}_{y \sim p(\cdot | s)} f(y, s) + \log \sum_x q(x) e^{f(x, s)}. \quad (6)$$

Writing $g(x) = q(x) e^{f(x, s)} / \sum_{x'} q(x') e^{f(x', s)}$, a distribution over candidates,

$$\mathcal{L}(s) = -\mathbb{E}_{y \sim p(\cdot | s)} \log g(y) + \mathbb{E}_{y \sim p(\cdot | s)} \log q(y), \quad (7)$$

a cross-entropy between $p(\cdot | s)$ and g plus a term that does not depend on the model. It is minimized at $g = p(\cdot | s)$, that is, at

$$f(x, s) = \log p(x | s) - \log q(x) + c(s), \quad (8)$$

the known optimum of sampled softmax [7, 58, 110]. With batch candidates, $q = p$ and the optimum is $\log \frac{p(x | s)}{p(x)} + c(s)$: how much more likely an item is after this history than on average. The contextualizer still learns everything that depends on the history, but popularity is divided out, and nothing in the loss rewards ranking a popular item above an unpopular one with the same lift. The fixed case is the special case of an uninformative history, $p(x | s) = p(x)$. The result is exact only as $K \rightarrow \infty$, assumes the model can represent the optimum, and is approximate when the batch holds duplicates.

Therefore, we also have a loss that we call the global loss, whose candidates are sampled uniformly from the full candidate space. With q uniform, the optimum is $f(x, s) = \log p(x | s) + c(s)$, which keeps popularity.

Note, we should not have both the distributions in one single loss. Meaning that we should not have

$$-\log \frac{e^{y_1^\top h}}{\sum_{x_i \in D_1} e^{x_i^\top h} + \sum_{x_i \in D_2} e^{x_i^\top h}} \quad (9)$$

because we are comparing across them: the model learns to identify, contrast, and separate the distributions, as the next paragraphs illustrate.

Instead consider:

$$-\log \frac{e^{y_1^\top h}}{\sum_{x_i \in D_1} e^{x_i^\top h}} - \log \frac{e^{y_2^\top h}}{\sum_{x_i \in D_2} e^{x_i^\top h}} \quad (10)$$

which is optimized by $p_{D_1}(y)$ and $p_{D_2}(y)$ respectively, with the two losses weighted by how often we sample from each distribution. This is better for us, because we want to learn both distributions individually rather than learning to contrast them: we care about generalizing to more data, not about identifying which dataset an example came from and separating them.

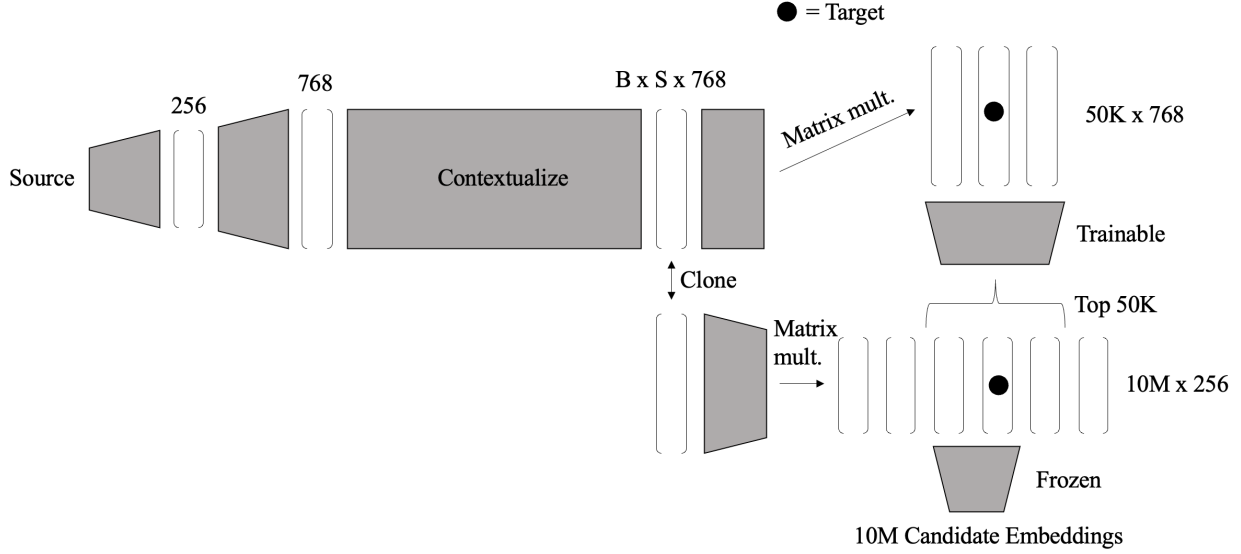


Figure 17: Hierarchical global–local candidate-scoring network, a design we have experimented with. The first stage uses a frozen 256-dimensional target embedder to project all 10 million actions, so a cheap matrix multiplication yields a coarse probability over the entire action space. The top-50k actions from this global distribution are then forwarded to a second stage, where a trainable 768-dimensional embedder refines the scores with a full softmax. Separating global screening from local re-ranking cuts computation and memory, while the larger embedder keeps full capacity for the final ranking.

Separate distributions. Assume two distributions p_{USA} and p_{Europe} with similar behavior and similar products, except that events in one are described in English and in the other in German. Trained on both together, the model learns to separate them easily on exactly this cue: it pushes the embeddings of US and European products far apart even when the behavior involving them is similar. In most cases this is not what we want. Deployments typically have separate output spaces that handle this anyway, so the main objective is to learn and generalize as much as possible. If we take such a model and ask how a US product will perform on the European market, it must project from the US space to the European space—two spaces it has been incentivized to separate—which is prone to error. If instead a second loss treats p_{USA} and p_{Europe} independently, there is less incentive to separate them and the spaces can align; adding correspondences between the assortments aligns them further, letting us more robustly ask how an event in one market corresponds to outcomes in another. The same holds across time, e.g. p_{2024} and p_{2025} : we are more interested in how events in 2024 help predict what happens in 2025 than in learning what happened when—we want to align rather than separate those events. We would not want to learn that nobody buys candy in 2025 just because it was removed from the assortment, rather than because users stopped buying it.

We tried restricting targets to only products bought that month and did not see a clear improvement. However, appropriate time domains did allow the model to do better.

As a guiding principle, the loss should compare only actions the user could have considered; anything else gives a false signal. This is tricky, because that space of possibilities can be very dynamic and unstable.

Global and local loss. Getting a global distribution over all actions and distinguishing within a local neighborhood may be two different bottlenecks, so the architecture can handle them differently. For example, with 10M possible actions, we can train stage 1 with two target embedders (the same could be done for the source, but this is more target-specific)—a smaller one with 256 output dimensions and a bigger one with 768—used as in Figure 17. The small embedder embeds all actions (it can be frozen and its outputs cached), enabling a much cheaper matrix multiplication in the smaller space that yields a distribution over all products. We then select the top 50K candidates and compute the softmax with the bigger embedder (which can be kept trainable) only on those; if the target is not among the top 50K we can skip the example or apply some other loss. This separates local from global and gives a hierarchical architecture with more efficient compute and parameter use.

History hierarchical loss. Since people often repeat actions they have already taken, we can create a history-hierarchical loss that factorizes into $P(Action) = P(Action|Action \text{ in History}) \times P(Action \text{ in History})$. This makes it easier for

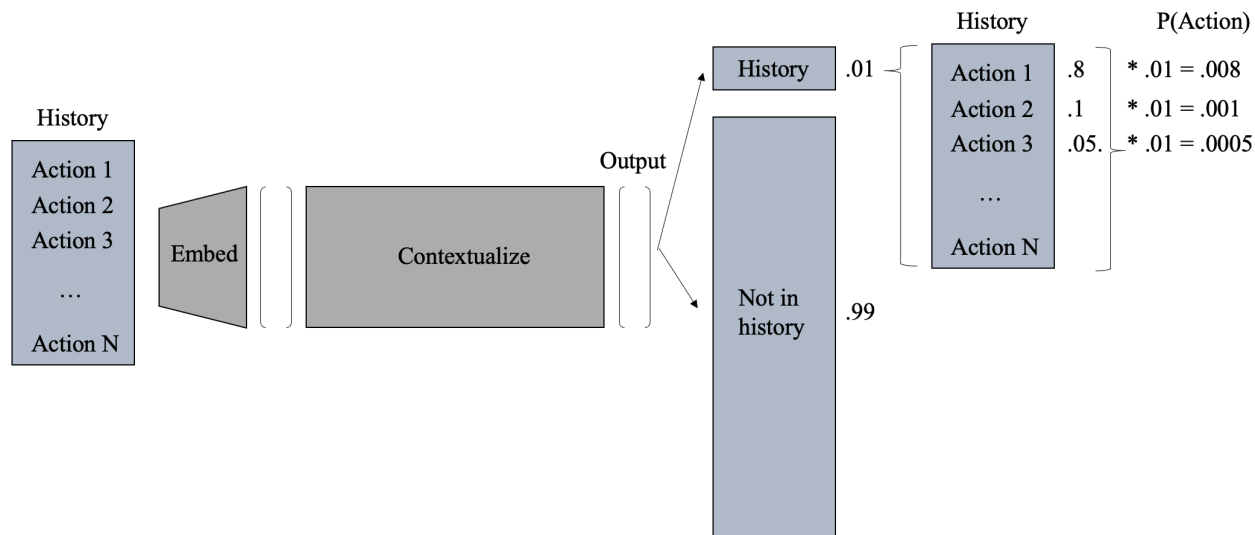


Figure 18: History-hierarchical loss schematic. The network factorizes the action probability into two sequential predictions: (1) a history gate that estimates the likelihood an action has occurred before, $P(\text{Action in History})$, and (2) a conditional scorer that assigns $P(\text{Action} \mid \text{Action in History})$ to each previously seen action. Their product forms the full distribution $P(\text{Action})$, biasing the model toward repeats while still allowing exploration.

the model to retrieve from history and learn the distribution of behavior, because we enforce a stronger bias. However, we could also do this with thresholding as in Figure 17, and hopefully learn the correct conditioning.

Collisions. Mapping the same event or action to one token across datasets aligns their spaces, but pre-defined mappings always introduce some incorrect correspondences, so the question is how much a correct correspondence helps and how much an incorrect one hurts.

MLM vs CLM objective. We have experimented extensively with versions of both Masked Language Modeling (MLM) and Causal Language Modeling (CLM) approaches for behavior. One difference from language modeling is the utility of predicting backwards. For example, given a product, we might want to ask what search term would be most likely to precede it, i.e. $p(\text{search}_{n-1} \mid \text{product}_n)$, which is different from predicting what search a user might do following the product. MLM-like objectives can capture this bi-directional conditioning [108]. However, we find that at scale MLM converges much slower than CLM and ends up working less well. Instead, we do our main training via CLM, then finetune specific capabilities efficiently afterwards using MLM-inspired objectives. This is quite analogous to findings for LLMs.

B.6 Fine-tuning Techniques

B.6.1 Exponential Moving Average (EMA)

We explored using EMA for finetuning, so that the model stays closer to the pre-trained one: stage 2 was updated as an EMA of the stage-1 model. This helped slightly but was not worth the complexity at scale.

B.6.2 Distillation (Teacher and Student)

We tried distillation with the stage-1 model as a teacher, using the loss $0.1 \times \text{stage-2 loss} + 0.9 \times \text{teacher loss}$, where the teacher loss targets the scores from our stage-1 model, which exhibits much less diversity. This did reduce diversity, but at the expense of test and validation losses. We also applied it to search queries only: again much less diverse, but with higher validation and test loss for search. Evidently, our test and validation sets reward diversity.

B.6.3 Training and Inference Correspondence

Formats at inference must match training formats exactly to avoid confusing the model; training with extensive augmentations makes the model robust to the remaining variations. LoRA adapters work well here, keeping the last layer freely trainable (Table 49).

B.7 Categories and Themes

Categories and thematic breakdowns of actions, such as search refinements or product categories, help users explore the action space efficiently, and they need to be personalized to the user’s behavior. The right breakdown depends on intent: is the user looking for keyboards for gaming, piano, or writing? Knowing this lets us predict the categories most helpful to the user, those whose breakdown carries the most information (high entropy). A fixed set of categories therefore falls short, in both its headings and their probabilities; both need to be generated and personalized on the fly.

We want to model $p(\text{category}|\text{behavior})$ and be able to sample from it, but we also want to model $p(\text{action, category}|\text{behavior})$. This allows us to generate and rank the most important categories for the user, and to assign actions by

$$\text{action} \in \operatorname{argmax}_{\text{category}} p(\text{action, category}|\text{behavior}) \quad (11)$$

This needs to be done with very low latency. It is hard for LLMs to approximate these probabilities in a personalized and efficient way: without detailed user data it is hard to predict $p(\text{category}|\text{behavior})$, and harder still the more subtle $p(\text{action, category}|\text{behavior})$ that determines which action to assign to which category, which can change substantially based on the behavior.

Labelling from search behavior. The vocabulary for such breakdowns can be read off the model itself. For each product we rank search terms by $p(\text{term} | \text{product}) \propto p(\text{product} | \text{term}) p(\text{term})$, scoring each term as a one-event prompt and weighting it by its observed frequency. A group of products is then named by a vote over its members’ top terms, so the labels come from how users search for the products rather than from how the catalog describes them.

B.8 Reasoning in the Behavior Space and Test-Time Compute

This section frames reasoning and test-time compute in terms of user behavior. *Reasoning* is something of a misnomer here; it is better understood as the strategic use of test-time compute, exploration, and simulation. In language-based tasks, reasoning is often explicitly structured as `clear-task` $\rightarrow$ `reasoning` $\rightarrow$ `result`.

In behavioral settings, analogous structures arise more naturally. For example, in e-commerce, a typical interaction might be `search` $\rightarrow$ `viewing products` $\rightarrow$ `add-to-cart`, where the `viewing products` stage can be seen as an exploratory reasoning phase preceding the final decision.

Our objective is to design a framework in which reasoning and test-time computation are applied more universally. We achieve this through the following procedure:

1. From each training sequence, remove the final event, yielding a pair: `{user-history-except-last-action}`, `{last-action}`.
2. Train an autoregressive model by passing `{user-history-except-last-action <think>}` and letting it generate K candidate actions. Construct sequences of the form: `{user-history-except-last-action <think> {K-generated-products} last-action}` for training.
3. Perform supervised fine-tuning (SFT) with `{last-action}` as the target, using standard cross-entropy loss while masking all tokens except the final action.
4. At this stage, the model learns to largely ignore the reasoning trace following the `<think>` token, understanding the format and producing the correct action at the $K + 1$ position regardless of the reasoning sequence.
5. Apply GRPO by providing `{user-history-except-last-action <think>}` as input and assigning rewards to improve the quality of the reasoning trace with target `{last-action}`, thereby making test-time computation genuinely beneficial.
6. Compare performance between the GRPO-enhanced model with `<think>` and the base autoregressive model, to evaluate whether test-time computation improves outcomes. Importantly, even when ignoring the reasoning trace, performance should be no worse than the baseline.

Small variations of this procedure are also feasible; it yields a more flexible model that can allocate test-time compute adaptively. For example, when the `<think>` token is used, it performs explicit reasoning; otherwise, it can produce direct predictions. Note that this is a property of one set of weights rather than of a separate reasoning model: instruction-following and reasoning behaviors are folded into the same released checkpoint and selected at inference through the prompt format, so there is a single model to serve, evaluate, and reason about.

Lastly, we explored introducing residual connections across time steps. If the goal is to improve a baseline next-step prediction using test-time computation, it may be advantageous to propagate information from the baseline prediction

directly to the final step. Concretely, this can be achieved by adding (or preserving) the embedding or probability distribution of the baseline next-step prediction at the final stage. This allows the model to focus on refining or adjusting an existing prediction rather than reconstructing it from scratch.

C Data

C.1 Demographics and Seasonality

Behavioral prediction requires explicit demographic and temporal covariates (e.g., age, calendar date), where seasonality and cohort effects are first-order drivers of outcomes. This stands in contrast to general-purpose LLMs: while they can be prompted with temporal context, they are not inherently trained to model calendar time or domain identifiers, and naïvely encoding exact timestamps risks *domain overfitting*—memorizing contemporaneous events rather than learning patterns that generalize to future periods. Consider a product introduced at the end of the training window (e.g., an iPhone 16): the model must decide whether subsequent demand reflects a persistent novelty effect or a recurring seasonal spike, a distinction that is underdetermined with limited post-launch data. The challenge is sharper for genuinely unseen concepts (e.g., “Apple Intelligence”), which appear frequently in retail and other behavioral domains, inducing rapid concept drift akin to accelerated semantic change. To mitigate this, we combine coarser, semantically meaningful time features with dynamic behavioral signals: we embed time-since-event and popularity trajectories (Appendix B.1) and update the model as fresh interaction data arrive. This improves temporal generalization and reduces overfitting to specific dates.

C.2 Data Considerations

Volume alone does not make behavioral data learnable. To uncover a behavioral “language” in which patterns can be modeled and predicted, the data must also carry meaningful, structured signal. This section sets out the criteria we use to identify such data.

C.2.1 Meaningful and Predictive Signals

A foundation model can only learn from data if that data contains inherent structure. A model can be fed large volumes of data and left to identify patterns, but meaningful signal must be present for learning to occur. Pure noise offers nothing to compress or predict. If the future is statistically independent of the past (i.e., $p(\text{future}|\text{history}) = p(\text{future})$), no model can uncover useful patterns. Thus, we focus on behavioral events that are consistent indicators of individual traits—actions that reveal preferences, motivations, and likely future behavior.

We find that survey responses and self-reported forms, while sometimes useful for diagnostics, are often poor predictors of real-world behavior due to their inconsistency and low signal-to-noise ratio. In contrast, logged behavioral events—that is, how people spend time, take breaks, change schedules—are more reliable and predictive.

C.2.2 Volume and Variety of Events

For a foundation model to generalize, it must observe enough high-quality examples. We need tens of millions of relevant historical sequences where future behavior can be predicted from past activity; BehaviorGPT-v4’s corpus holds close to 2 billion unique sessions. Crucially, these sequences must contain consistent event types. If each user exhibits only unique types of events (e.g., one user only takes breaks, another only has sick days), the model struggles to learn generalizable patterns across employees.

It is more valuable to have co-occurring and overlapping event types (e.g., how breaks and sick days might jointly predict vacation time) than to have disjoint sets of behaviors per user. In short, we aim to capture joint probabilities across event types, not isolated patterns.

C.2.3 Depth Over Breadth (Longitudinal)

We benefit more from deeply understanding a few individuals than from shallow glimpses of many. Long sequences of meaningful events per individual enable the model to infer latent traits and behavioral tendencies, which are foundational for making accurate predictions about unseen users. Tracking should be per individual and as longitudinal as possible: aggregates lose the very structure we need; the goal is to be an anthropologist, not a statistician.

Longer sequences also connect concepts that are far apart in the data. A customer might add a vegan product and not add another one until 100 events later; only long contexts connect such recurring preferences, and through them more abstract associations (e.g., vegan customers also tending to buy outdoor food). Similarly, brand loyalty often reveals

itself only over long horizons: someone who refills the same brand of toilet paper monthly may require a 500-event context window before that pattern is even visible.

C.2.4 Analogies in Human Natural Language

Human natural language exhibits all of these desirable properties. Because language was developed as a tool for communication, we know it is not random; it carries structured, meaningful information, and it is inherently learnable (as evidenced by our ability to acquire it). Language is composed from a finite set of symbols, such as words or characters, meaning that sequences of text share common elements that can act as building blocks for learning. Moreover, we have access to long, coherent sequences produced by individuals (such as books), which provide rich insight into what the author intends to communicate.

Human behavior shares the learnability property: people can interpret a large part of each other’s behavior, which makes sense because it is difficult to produce behavior much more complicated than what we ourselves can interpret, consciously or subconsciously. The parameters on which we base our behavior and those on which we base our interpretation of behavior are related.

C.3 Workforce Events

The recipe was also applied, separately, to workforce data [34]; we summarize that study here because Section 2.6 and the Discussion rely on it. The corpus holds 43M weekly events from about 80,000 employees over four years, drawn from 15 event types (scheduled, actual and productive hours, paid and unpaid breaks, and similar) with eight contextual attributes such as manager and campaign. Identifiers were salted and timestamps jittered, and training, validation and test sets are split by employee, so every test sequence belongs to an unseen person. A 5M-parameter, eight-layer causal transformer is pretrained from scratch with next-event prediction, then fine-tuned at every prefix to predict whether, when and why an employee leaves. On 6,705 held-out employees (4,653 leave within the month, 2,052 stay), each truncated at a random point and asked whether they will leave within the next month, the model reaches 91% accuracy and F1 0.93, with 99.3% specificity and 0.997 precision. Starting from the pretrained weights improves attrition performance by 7% over training the same architecture from scratch. Mean-pooled employee embeddings cluster by country, with weaker, distributed structure by tenure and reason for leaving. These are results of a different, far smaller model on a different domain: they show that the architecture and objective transfer, not the BehaviorGPT-v4 checkpoint.

C.4 Text and Internet Data

One way to incorporate more general text data is to segment it. Our approach retrieves the next actions (described by text, images, and/or numbers) given a sequence of such historical actions. Thus, we take a text sequence that is topical, perhaps about electronics products or receipts, and we randomly segment it into segments of, say, 5–25 words, and just predict the next segment based on previous segments.

Prompting may also help as the model is trained on more scraped data. Given the products a user has bought, we can append a text segment that states the task, differently for search and for recommendation:

```
Search:      [product 1] [product 2] [The user bought these items and now searches for
'milk']. The next product is]
Recommendation:      [product 1] [product 2] [The user bought these items. The next
product is]
```

Any such reasonable prompt can be constructed. See Figure 19 for an example in Swedish (“fisk” means fish), where this worked well. This lets us utilize topical text corpora as a lowest-quality tier of data that is cheap to obtain.

Similar constructions apply to other data, such as videos: crop at certain frames, embed with an image embedder, then treat the result as a sequence and apply contrastive learning on the embeddings, i.e., classify the correct “next” frame-crop against random frame-crops. This should learn how things relate “causally”.

C.5 Acquiring Behavioral Data Across Channels

Behavioral data can also be amassed by *acting*: interacting with e-commerce sites directly to elicit more data. BehaviorGPT can rank alternatives and even generate search queries, so we can let a strong model sample short start-sequences (1–5 products), follow similar/related-product links, or issue generated queries and learn from the search suggestions. Training on the collected traces, then generating more data with the improved model, and iterating in an expectation–

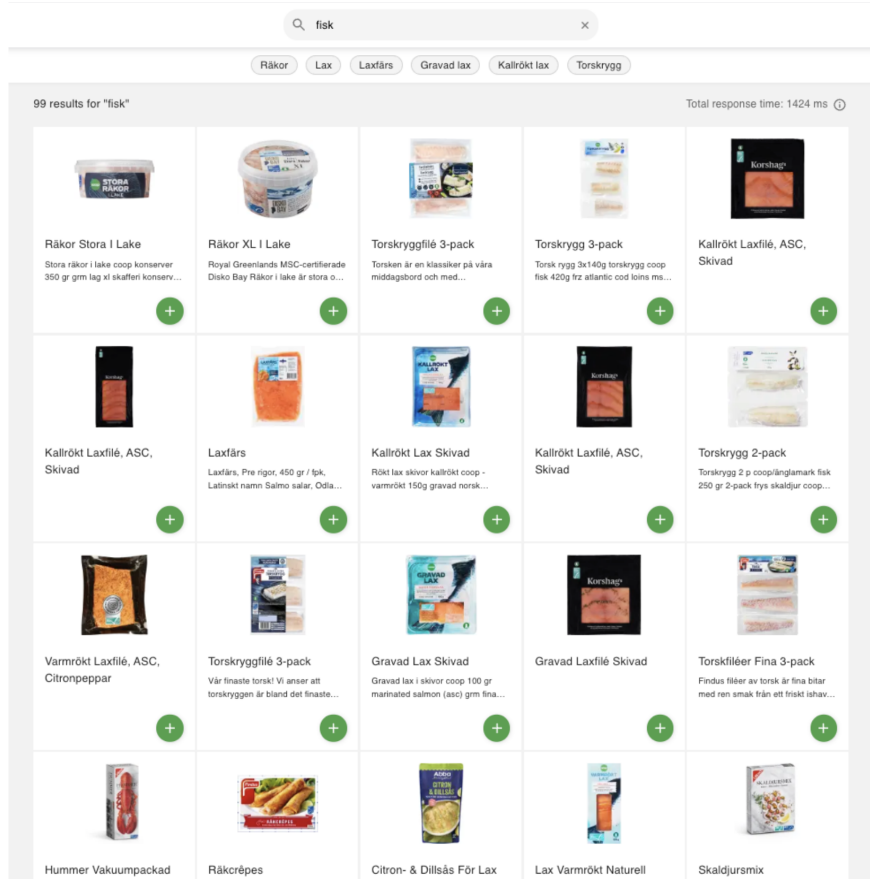


Figure 19: Using a sequence segmentation on topical internet text to get a starting point for a contextual retrieval model.

maximization fashion, converges toward a compressed version of the target site’s algorithms—which we can then improve upon. A large model can generate this data, which could then be distilled into smaller models.

C.6 Data Augmentations

Data augmentation is nearly absent from current LLM training, especially in pretraining. The abundance of data is the main reason, but the task also contributes. In language modeling, we want to be able to predict the next token given the previous tokens. However, in behavior, we typically want to predict any future action based on any previous action. If a person has bought a cheese burger we might want to understand what type of car the person might be interested in purchasing, without having a natural sequence of events between them. Modeling such links with long beam searches or similar methods can be suboptimal and practically impossible. That is, the $p(\text{future}|\text{history})$ in behavior is much less connected and restricted than it is for LLMs, where we want the next immediate tokens.

Because behavior is tracked, it also depends strongly on the tracking system that recorded it, which we do not want. We aim for a representation that does not overfit to any one tracking system and works for any format of behavioral sequence, and augmentations help here too.

Also, if we drop certain features in the right way—so that they are used both as supplements and as complements at a given part of the network—we can generalize better and also learn embedding spaces that are semantically meaningful, allowing different features to be embedded into a shared space. This is useful for many downstream tasks, and it augments the text-and-image correspondences with sequence information: an image is embedded by how it appears in sequence with other images and text, not just by the description of the same product, which reduces overfitting and gives the space a more behavioral meaning.

Consider an image–text pair: it has a single relation. Add some type of augmentation, with n possible augmentations of each side, and we get $n \times n$ relations (such augmentations are usually automatic rather than user-defined, and thus less

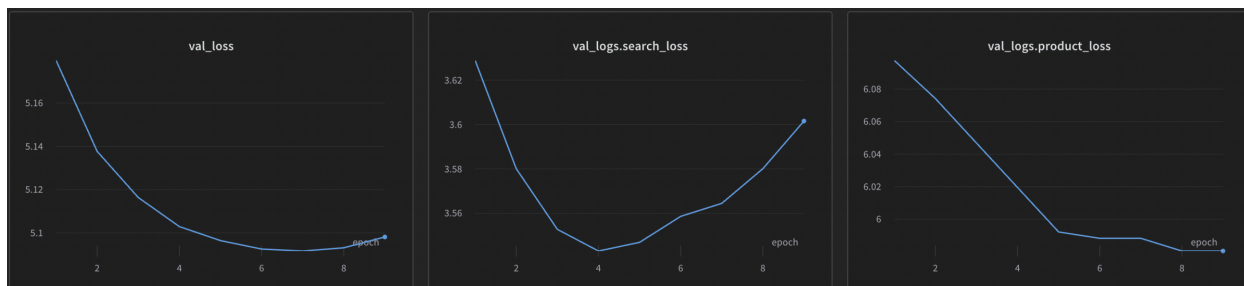


Figure 20: Tasks converge at different rates

meaningful or genuine). Now consider a sequence of k such pairs: each instance introduces $(n \times n)^k$ relations, many of which are defined by the actual sequential behavior of the user.

Augmentations are also critical because the model should be promptable flexibly, without expecting a specific format that makes it harder to use and debug—for example, a model that comes to require prompts ending with a period “.” because its training data was created a certain way.

Where to make use of LLMs. We find that LLMs are not competitive for contextualizing user behavior. However, they are useful at the per-event level: their general knowledge is best incorporated per event rather than per individual user. We use an LLM to generate metadata for events and to perform basic sanity checks. This way the LLM only needs to be called once per unique event in the vocabulary, rather than once per user-event in a history—the latter being several orders of magnitude more expensive. LLM-generated item descriptions used as augmentations improved generalization (Appendix G.7).

C.7 Missing Critical Actions

When critical actions are missing from the data, the model finds causation where there is none and cannot identify the factors that actually matter—so a better loss does not translate into better real-world performance.

One example of many: our search was becoming a little too diverse—jumping ahead too much—according to our own qualitative evaluations, even while improving in cross-entropy loss and top-k accuracy. The shortfall therefore lay in the data rather than in the learning process: the model learned what the data showed. Digging deeper into the tracking, we found a way to more reliably record whether a user who searched then navigated away from search to a new page and clicked something there. Including this new action in training let the model learn much more clearly whether a search led to a direct click or the user navigated away, and made search markedly less diverse. Still, this had a smaller effect on A/B test performance than expected: the diversity was more annoying for analytical testers than for real users, since the results, though diverse, were mostly relevant.

The back and forth between data and performance, qualitative and quantitative, is key to finding problems with the data and to optimizing performance.

D Training, Systems, and Metrics

D.1 Tasks Converge at Different Rates

Predicting next actions is more explicitly multi-task than language modeling: a single training sequence supervises search, recommendation, and other behaviors simultaneously, and some of these tasks are inherently easier than others. Search, for instance, is target prediction given extra information (the query), and its loss converges much faster than product-to-product recommendation (Figure 20), so search may begin to overfit while recommendation is still improving, raising the question of where to stop early. We initially addressed this by regularizing fast-converging tasks more heavily so that tasks converge at comparable times. As dataset sizes grew this became less of an issue, and the dominant concern shifted to not overfitting on any specific domain or data source.

D.2 Training Stability

Reordering normalization layers is important for stabilizing training of bigger models and across data sources, in the embedder as well as the contextualizer. We normalize the output of each sub-layer before it joins the residual stream,

and never normalize the stream inside a block: each block computes $x + \text{Norm}(\text{Attn}(x))$ and $x + \text{Norm}(\text{MLP}(x))$, with RMSNorm and query–key normalization throughout. After the last block we apply RMSNorm, project to the 768-d output space, and normalize the result to unit length; candidate embeddings are unit-normalized too, and a learned temperature scales the cosine logits. In one earlier run a candidate table collapsed, which we now catch early by logging embedding norms and pairwise cosines. Batches with a non-finite loss are skipped on all devices and saved, with their sampled candidates, for inspection.

D.3 GPU Utilization

In stage 2 the candidate table is the dominant memory cost. For a catalog of 710M items, 768-dimensional bf16 embeddings occupy 1.07 TB, or 134 GB per GPU if sharded across the eight GPUs of a node, which does not fit. We therefore hold the full table once per node in host memory, as a memory-mapped file that all eight ranks share through the operating system’s page cache, so the node pays for one copy rather than eight. At every step each GPU samples its negatives without replacement from the full table, reads only those rows from host memory and moves them to the device. The per-GPU samples are then gathered, so every query is scored against the negatives of all devices in a single softmax; Table 47 shows why one joint softmax beats one per device. A background thread samples and gathers the next step’s negatives into pinned buffers while the current step computes, which keeps the lookup off the critical path. Where a catalog is small enough, the table can instead be sharded row-wise across GPUs, each GPU embedding the ids it holds and exchanging the rest; this was our earlier setup at 64M items. In both cases the softmax is aggregated across GPUs by max- and sum-reductions of the streaming statistics of Appendix D.4.

D.4 A Custom Softmax for Huge Vocabularies

Vocabulary size is a particular challenge in our setting. Across the whole corpus the action space exceeds 10 billion unique events, which no exact softmax can reach; even a single catalog runs to the order of 10^9 items, roughly ten thousand times a typical LLM vocabulary, which already makes materializing the full logit matrix impossible under single-GPU memory constraints. Sampling therefore governs pretraining (Appendix B.5), but wherever a full catalog has to be scored we score it *exactly*, at V on the order of 10^9 . The underlying technique is online softmax [80], as used in fused large-vocabulary cross-entropy kernels [102]; what we add is an optimized streaming kernel for our distributed setting: we algebraically merge several operations to avoid ever materializing the full $(B \times V)$ matrix, implement a streaming log-sum-exp over vocabulary shards, fold in the time and sales residual heads, and derive a custom backward pass with hand-computed gradients for the fused operations. We give the construction below for completeness.

Setup. Let B be the batch size, H the hidden dimension, and V the number of candidates. For each example $b \in \{1, \dots, B\}$ we have a hidden vector $h_b \in \mathbb{R}^H$ and a target index $y_b \in \{1, \dots, V\}$. Let $W \in \mathbb{R}^{V \times H}$ denote the candidate matrix with rows $w_j^\top$. We optionally use two additive “residual” heads: (i) a time–difference (“timedelta”) head with embedding matrix $E^{(\text{time})} \in \mathbb{R}^{T \times H}$ and per-example logits $g_b^{(\text{time})} = E^{(\text{time})} h_b \in \mathbb{R}^T$, and (ii) a sales–since head with embedding matrix $E^{(\text{sales})} \in \mathbb{R}^{S \times H}$ and logits $g_b^{(\text{sales})} = E^{(\text{sales})} h_b \in \mathbb{R}^S$. Each candidate j is assigned indices $\tau_{bj} \in \{1, \dots, T\}$ and $\sigma_{bj} \in \{1, \dots, S\}$ (e.g., bucketed by time delta and sales history for example b). Let $m_{bj} \in \{0, 1\}$ be a mask indicating if candidate j is *invalid* for example b (type or time availability); we always enforce $m_{b, y_b} = 0$.

Logits with residuals and masking. For each (b, j) the (pre-masked) logit is

$$\tilde{\ell}_{bj} = \underbrace{\langle h_b, w_j \rangle}_{\text{linear decoder}} + \underbrace{g_{b, \tau_{bj}}^{(\text{time})}}_{\text{timedelta residual}} + \underbrace{g_{b, \sigma_{bj}}^{(\text{sales})}}_{\text{sales residual}}.$$

Masking is applied as

$$\ell_{bj} = \begin{cases} \tilde{\ell}_{bj}, & \text{if } m_{bj} = 0, \\ -\infty, & \text{if } m_{bj} = 1, \end{cases} \quad \text{with } m_{b, y_b} = 0 \text{ by construction.}$$

Shardwise softmax with streaming log-sum-exp. To avoid materializing $(B \times V)$ logits, we partition the vocabulary into S contiguous shards $\{\mathcal{S}_s\}_{s=1}^S$ of size at most K (the `slice_size`). We process shards sequentially while maintaining, for each b , a running maximum M_b and a running softmax denominator Z_b :

$$M_b^{(0)} = -\infty, \quad Z_b^{(0)} = 0.$$

For shard s we compute local logits $\{\ell_{bj}\}_{j \in \mathcal{S}_s}$ and set

$$m_b^{(s)} = \max_{j \in \mathcal{S}_s} \ell_{bj}, \quad \hat{M}_b = \max\{M_b^{(s-1)}, m_b^{(s)}\}, \quad \alpha_b = \exp(M_b^{(s-1)} - \hat{M}_b).$$

We then update

$$Z_b^{(s)} = \alpha_b Z_b^{(s-1)} + \sum_{j \in \mathcal{S}_s} \exp(\ell_{bj} - \hat{M}_b), \quad M_b^{(s)} = \hat{M}_b.$$

Let T_b denote the (unshifted) target logit, $T_b = \ell_{b, y_b}$, which we record when the shard containing y_b is processed. After the final shard,

$$\text{loss}_b = \log Z_b^{(S)} - (T_b - M_b^{(S)}) = -\log \frac{\exp(T_b)}{\sum_{j=1}^V \exp(\ell_{bj})}.$$

The total loss is reduced as desired (mean/sum/none).

Probabilities and gradients. Define the exact softmax probabilities (implicitly recoverable shardwise)

$$p_{bj} = \frac{\exp(\ell_{bj} - M_b^{(S)})}{Z_b^{(S)}} \quad (\text{with } p_{bj} = 0 \text{ if } m_{bj} = 1).$$

Let $\delta_{bj} = \mathbb{I}\{j = y_b\}$ and define the standard softmax residual $\gamma_{bj} := p_{bj} - \delta_{bj}$. Then the gradients are:

$$\frac{\partial \text{loss}}{\partial w_j} = \sum_{j=1}^V \underbrace{\gamma_{bj} w_j}_{\text{decoder term}} + \underbrace{\sum_{t=1}^T \left(\sum_{j=1}^V \gamma_{bj} \mathbb{I}\{\tau_{bj} = t\} \right) e_t^{(\text{time})}}_{\text{timedelta head}} + \underbrace{\sum_{s=1}^S \left(\sum_{j=1}^V \gamma_{bj} \mathbb{I}\{\sigma_{bj} = s\} \right) e_s^{(\text{sales})}}_{\text{sales head}},$$

where $e_t^{(\text{time})}$ and $e_s^{(\text{sales})}$ are the t -th and s -th rows of $E^{(\text{time})}$ and $E^{(\text{sales})}$, respectively. For the residual embedding matrices,

$$\begin{aligned} \frac{\partial \text{loss}}{\partial E^{(\text{time})}} &= \sum_{b=1}^B \left(\rho_b^{(\text{time})} h_b^\top \right), \quad \rho_{b,t}^{(\text{time})} = \sum_{j=1}^V \gamma_{bj} \mathbb{I}\{\tau_{bj} = t\}, \\ \frac{\partial \text{loss}}{\partial E^{(\text{sales})}} &= \sum_{b=1}^B \left(\rho_b^{(\text{sales})} h_b^\top \right), \quad \rho_{b,s}^{(\text{sales})} = \sum_{j=1}^V \gamma_{bj} \mathbb{I}\{\sigma_{bj} = s\}. \end{aligned}$$

Implementation details (streaming). The quantities $M_b^{(s)}$ and $Z_b^{(s)}$ are maintained per batch row while iterating shards, using the numerically stable log-sum-exp updates above. During the forward pass we also accumulate the sufficient statistics needed to form $\partial \text{loss} / \partial h_b$ without storing full $(B \times V)$ tensors: for each shard we add $\sum_{j \in \mathcal{S}_s} \exp(\ell_{bj} - M_b^{(s)}) w_j$ to a running buffer, which is first rescaled by α_b whenever the running maximum changes, exactly as Z_b is, and is divided by $Z_b^{(S)}$ and corrected by subtracting w_{y_b} at the end (realizing $\sum_j p_{bj} w_j - w_{y_b}$). Analogous per-index scatter-add accumulators, rescaled by α_b in the same way, realize $\rho_b^{(\text{time})}$ and $\rho_b^{(\text{sales})}$. The gradient w.r.t. W is formed by a second shardwise pass that reconstructs p_{bj} from $(M_b^{(S)}, Z_b^{(S)})$.

Complexity. The method computes the *exact* softmax cross-entropy while using $O(BH + KH)$ working memory per shard (instead of $O(BV)$), with total compute matching a dense pass up to streaming overhead.

Measured cost. At production stage-2 shapes on one B200 GPU (8,192 query positions, from 16 sequences of 512 events, against 262,144 sampled candidates, with the time, sales and type terms enabled), materializing the logits alone would take 4.3 GB in bf16, and as much again for their gradient. The streaming kernel runs the full loss step, forward and backward, at a peak of 2.6 GiB. Compiling the per-shard body and prefetching the candidates on a background thread cut the step from 228.8 ms to 57.5 ms, a $4.0 \times$ speedup. The loss agrees with the unfused implementation to 1.4×10^{-5} , and the gradients sit closer to an fp32 reference than the eager bf16 loop does.

D.5 Losses That Depend on Sampled Candidates

Our loss depends on the sampled candidates, not only the input batch. Inspecting the batch behind a loss spike therefore requires the candidate space as well, so we save it, which adds overhead.

For the same reason, comparing models across training runs with different candidate sampling needs a separate metric: a subset of the logits, an accuracy, or the validation or test evaluation run on the same fixed candidate space.

D.6 Cart Metric

In basket-completion, the exact order of forthcoming items is often immaterial—predicting *Fanta* instead of *Coca-Cola* is still far more useful than predicting *Salmon*, yet standard cross-entropy penalizes both mistakes equally. Unlike language modeling, where order matters, cart prediction benefits from a loss that tolerates permuted order. A sufficiently large model could in theory learn this from data alone, but we make it explicit with **CartMetric**. Given past carts $(c_1, \dots, c_{k-1})$ and the subset of items already placed in the current cart $c'_k \subset c_k$, $\text{CartMetric}(n, (c_1, \dots, c_{k-1}), c'_k, c_k)$ measures the probability that the model’s top- n predictions intersect the unseen set $c_k \setminus c'_k$. Because it rewards any correct item regardless of position, the metric tracks real-world recommendation quality more closely.

Finally, we ensure that *each ground-truth product can contribute to the score at most once per cart*. Let $H_t \subseteq \hat{c}_t$ be the set of correct hits among the model’s top- n predictions at step t . We maintain a global hit set $H = \emptyset$ for the entire session and update $H \leftarrow H \cup (H_t \setminus H)$ after every step; only the $|H_t \setminus H|$ new hits are credited. Consequently, predicting *Fanta* correctly at multiple steps yields a single reward, forcing the model to surface *novel* items to improve its overall score and preventing precision inflation through repeated mentions of the same popular product.

E Public Datasets in Detail

Figure 1 averages the two datasets, and three of its claims are per-dataset claims: that the text-reading baselines are ordered by how much of an item its text describes, that on short video fine-tuning on the first thousand target samples drops the 12.5B slightly below its zero-shot score, which does not happen on reviews, and that the 0.5B is close to the 12.5B only where the target resembles our pretraining mixture. Figure 21 redraws the average with the spread between the two datasets shaded, and Figure 9 draws the same runs one dataset per row on absolute recall, which is where those three claims are read.

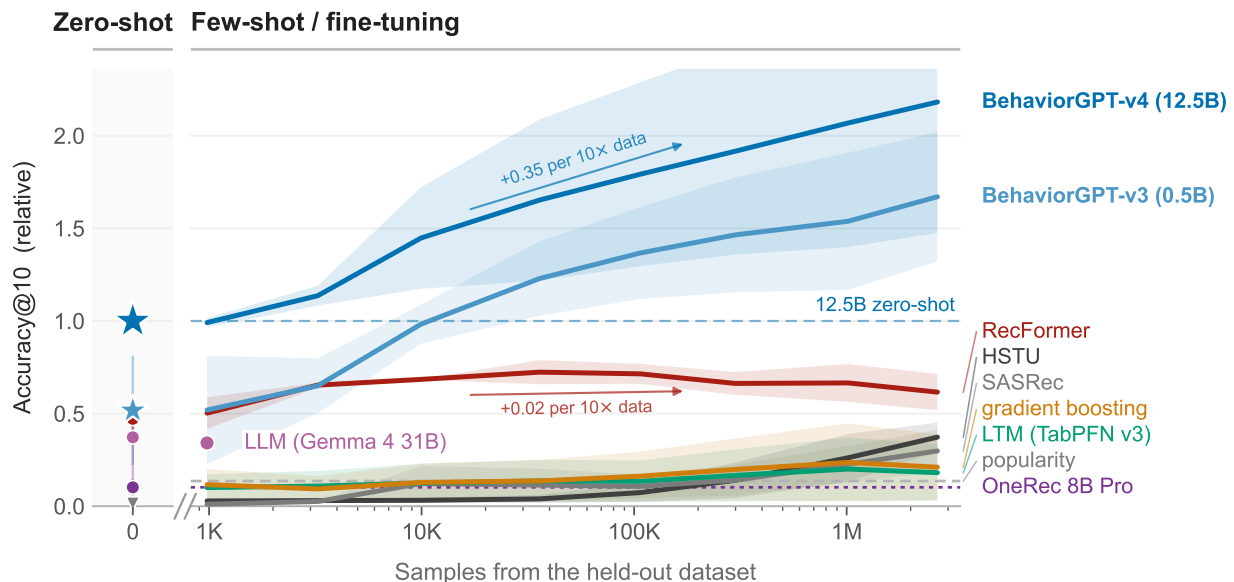


Figure 21: Figure 1 with the spread between the two datasets drawn: each band and each zero-shot bar runs from an arm’s lower to its higher normalized score across short video and reviews, around the arithmetic mean drawn as the line. Seed spread is far smaller and is not drawn.

How the average is formed. Each score is divided by the 12.5B’s zero-shot score on its dataset, 0.594% on short video and 0.763% on reviews, and the two are averaged arithmetically; normalizing first keeps the mean from being an average of dataset difficulty, the corpora differing by a factor of two. The sweeps use eight matched budget rungs (945/1013, 3219/3276, 9858/10040, ...), each averaged point being two measured runs drawn at the geometric mean of their sample counts, with nothing interpolated. Curves are 5-seed means; seed spread is 0.6–2.4% on the 12.5B, and zero-shot inference is deterministic. A target a method structurally cannot score counts as a miss, so arms confined to a bounded pool or to items seen in training are penalized as they would be in deployment: Gemma, LightGBM and the LTM rank a shortlist of the target’s most frequent products, and SASRec and HSTU reach only items in their training interactions. On reviews the two specialists rank within the 137,027 items of their training interactions, 8.6% of our catalog, which is much of why they sit so low there. Popularity is a rule on both rows so that they stay on one footing, counted per budget on reviews but over the full training stream on short video.

With no target data. LightGBM, the LTM (TabPFN) and popularity have no zero-shot point, having nothing to fit, condition on or count. Gemma has one, but not a target-blind one: it ranks the target’s most frequent products rather than its catalog. The sequential specialists land at chance, $0.009\times$ for SASRec and $0.004\times$ for HSTU, as a model that reaches items through learned identity must on an unseen catalog. The rest are ordered by how much of an item its text describes, and this is where averaging hides most: RecFormer’s $0.47\times$ is nearly the same on both datasets, while a prompted Gemma’s $0.37\times$ is low on short video, whose items are videos, and high on reviews, whose titles are descriptive. The 0.5B, at $0.52\times$, varies for the analogous reason, reviews being close to our pretraining mixture and short video not; whether it or Gemma leads depends on the choice of mean, so nothing rests on it. OneRec 8B Pro reaches $0.10\times$.

After adaptation. No baseline reaches the rule at any budget. RecFormer stops improving, peaking at $0.72\times$ by 36,000 samples and ending at $0.62\times$. We see two likely causes, neither isolated here. RecFormer is fine-tuned under its authors’ recipe, on leave-one-out splits of each user’s history, but scored on our temporal split with unseen users, so more target data fits it more closely to a task other than the one measured. And it represents items by text alone, where ours is multimodal, and both its architecture and the way it is trained differ markedly from ours; on these results, that approach appears less effective at converting target data through fine-tuning. Ours, meanwhile, rise at every rung, averaged over the two datasets, to $2.18\times$ for the 12.5B and $1.67\times$ for the 0.5B; per dataset, the 12.5B ends at $2.89\times$ on short video and $1.48\times$ on reviews, the two drawn under different fine-tuning protocols (below). The 0.5B crosses the rule between 9,949 and 35,928 samples, so about ten thousand target samples buy what $25\times$ the parameters supply for free. Popularity and OneRec are drawn flat: OneRec uses no target data, and popularity has per-rung counts only on reviews, so its rule takes reviews’ largest rung.

In absolute terms. Zero-shot, RecFormer and Gemma reach 0.402% and 0.465% on reviews against our 0.763%, but only 0.248% and 0.081% on short video against our 0.594%; OneRec reaches 0.087% and 0.044%. After the full sweep the 12.5B reaches 1.717% on short video and 1.126% on reviews. On reviews we name no strongest baseline after training, RecFormer and HSTU tying within a seed at 0.395% and 0.345%.

The two rows use different fine-tuning protocols. On short video the first thousand samples leave the 12.5B slightly *below* its zero-shot score, at $0.97\times$, recovered by 3,219 samples; on reviews it starts level at $1.02\times$ and rises at every budget. The difference comes from the protocol, not the dataset. On reviews the first adaptation stage trains only the last transformer block, final layernorm, output projection and domain embedder (0.20B of 12.52B), for eight epochs per stage. Restricted this way the curve rises at all nine budgets with 0.6–2.4% seed spread; unrestricted, it starts at $0.84\times$, dips at 298,374 samples and has 1.9–8.1% seed spread, though at the largest budget it reaches 1.231%, 8.6% above the 1.126% drawn. Each curve is drawn under one protocol end to end, the one that leads at most budgets on its dataset. On short video the restriction removes the dip ($1.05\times$ at 945 samples) but trails the full fine-tune at the other seven budgets, ending at $2.11\times$ against $2.89\times$; on the 0.5B it loses at seven of eight budgets on reviews.

Reference time on short video. The checkpoint scores an item by combining its content embedding with a residual read from the item’s age and its past sales, both measured at a reference date. Figure 1 sets that date per position, to the date of the position being scored, for 0.594% zero-shot recall@10; every number in this paper is read that way. Where one date must serve every position instead, recall@10 falls to 0.330–0.443% depending on where in PixelRec’s validation window (2021-11-13 to 2022-06-25) that date falls, highest mid-window and lowest at either end. Table 6 gives the sweep.

Why the hosted catalog carries one date. In production the item embeddings are precomputed and the residual is folded into each stored vector when the catalog is embedded, which is what lets one stored vector serve a request. Item fields change rarely and age is bucketed by month, so little is lost as long as catalogs are re-embedded regularly and the

Reference time	Future	R@1	R@10	R@100	NDCG@10	MRR@10
<i>Our harness, 89,444 candidates</i>						
Each position's own date	kept	0.091	0.594	3.346	0.293	0.203
Each position's own date	removed	0.113	0.777	4.035	0.382	0.264
2021-11-13 (window start)	kept	0.035	0.330	2.047	0.158	0.106
2021-11-27 (10% of positions)	kept	0.038	0.356	2.162	0.169	0.113
2021-12-20 (25%)	kept	0.040	0.398	2.572	0.184	0.120
2022-01-29 (median)	kept	0.062	0.432	2.684	0.213	0.147
2022-03-04 (calendar midpoint)	kept	0.062	0.433	2.581	0.212	0.146
2022-03-19 (75%)	kept	0.062	0.443	2.552	0.217	0.149
2022-06-25 (window end)	kept	0.049	0.365	1.981	0.178	0.122
<i>Public API, hosted catalog PixelRec200K, 89,437 products</i>						
2022-01-29 (fixed at embed time)	kept	0.060	0.398	2.283	0.201	0.142
2022-01-29 (fixed at embed time)	removed	0.132	0.953	5.285	0.470	0.325

Table 6: **What a single reference date costs, and what removing future candidates buys.** Zero-shot 12.5B on all 79,399 PixelRec validation positions whose answer is a product; every row scores the same positions, and all figures are percentages. *Reference time*: the date at which an item’s age and past sales are read. *Future*: whether items first seen after the position’s date stay in the candidate pool. The first row in **bold** is the protocol behind every PixelRec number in this paper. The hosted catalog holds the same products as our harness less the seven special tokens, and its date is fixed when the catalog is embedded, so the two *kept* rows at 2022-01-29 are the like-for-like pair, 0.432% against 0.398%.

reference date stays near the present. Stored vectors also allow vector search over the catalog, indexed with HNSW over int8-quantized vectors and rescored at full precision over the best candidates, which trades a little accuracy for speed on large catalogs. For a benchmark on historic sequences the consequence is that one embedding carries one date, and matching this paper exactly would need one embedding of the catalog per evaluated date, 225 of them across the validation window, which is not practical to host. The hosted catalog is therefore embedded at the median date of the evaluated positions, 2022-01-29, and Table 6 reports our own evaluation at that same single date beside what the public API returns on the same positions.

Candidates from the future. No number in this paper removes items first seen after the position being scored, so the model ranks against a catalog that is partly in that position’s future. It is the harder setting and we report it throughout. Both our harness and the API can remove those items instead, and doing so moves the result further than the reference date does, in the opposite direction: on each position’s own date recall@10 goes from 0.594% to 0.777%, and on the hosted catalog from 0.398% to 0.953%. A reproduction that turns the filter on will therefore exceed this paper’s figure rather than match it, which is why both settings are in the table.

What the notebook reproduces. We publish part of the harness: a preprocessing script that builds the split from the official PixelRec download, a notebook that scores every evaluated position through the public API and computes the metrics from the returned rankings, and the pre-embedded catalog, hosted for every API key. A run needs an API key and the two PixelRec files and covers all 79,399 positions, with a 5,000-position subset for a quicker check. Its reference date is the hosted catalog’s and its candidates are unfiltered, so the row it reproduces is the first public-API row of Table 6 and not the paper’s.

F The Breadth Matrices in Detail

F.1 How to Read the Matrices

Six grids share one layout—Figures 5 and 12 in the body, the second of them a pair of panels, and Figures 11, 25 and 22 on the same axes—and what changes between them is only what each cell divides. The conventions are set out once here rather than in five captions.

Rows. The 12 benchmark datasets, bracketed by industry into retail and engagement, with the payments block set apart below. *Engagement* is app usage: streams where a session browses, watches, rates or discovers rather than buys, which is what puts beverage discovery, image discovery, short video and a review corpus in one bracket. Payments is drawn apart because its two tasks predict an attribute of the next event—which merchant, which category—rather than the item itself, making them hit@5 rather than recall@10, and against a different pool: the strongest of twelve arms, led on both tasks by a rule that repeats what the session has just carried, ahead of one over the user’s own history, RecFormer fine-tuned and zero-shot, TabPFN v3, OneRec 8B Pro, Gemma 4 31B, a popularity rule, a rule that answers the five commonest labels, LightGBM, and SASRec and HSTU, which trail the rest. The two tabular arms predict the

label directly where every other arm in the pool, ours included, ranks products and reads the label off them; we lead both tasks at every k , at hit@1 by 63.0% against 53.3% on merchant and 35.9% against 30.0% on category (Appendix F.3). Short video and reviews are scored under Figure 1’s protocol rather than the benchmark’s, and are the two rows whose pool is publicly reproducible. Table 7 gives what the rows differ in.

Columns. Tasks, three of them selected by the event preceding the target: a full search query, a partial one, or a purchase. *View recall* pools mid-session views with session starts. *Item returned* is the one column selected by its target instead of by what precedes it—which item a user will return—so it overlaps the five columns beside it rather than extending them, and is scored over every target rather than over products alone. An empty cell is a task the dataset does not log rather than a run we skipped: four rows log purchases, seven log search. Every task here has the session holding something in front of the target. Positions with nothing in front of them at all ask a different question—what a prior over an unseen catalog is worth on its own, rather than what the model does with a user it can watch.

Marks. $\sim$ marks a baseline that scored a different number of positions than we did, its recall read as an estimate of what it would score on ours. z marks a cell showing the unadapted checkpoint where the figure otherwise draws a fine-tune. Marketplace’s five cells and payments’ two were not fine-tuned because both streams are close to the pretraining data, so the unadapted checkpoint is what we would deploy there; image discovery’s two were not because the unadapted checkpoint already led the whole pool, by $5.6\times$ and $3.7\times$. Each is evidence about the checkpoint and none about adaptation, so no aggregate over adaptation is computed from them, and in Figure 25, where the denominator is the zero-shot run itself, they are grey rather than carried: a run divided by itself would print a gain of $1.00\times$ on a cell that measures none. $+$ marks a cell where the 0.5B scored a narrower evaluation window than the 12.5B—marketplace, from 2.4% fewer positions on partial search to 53.5% fewer on the return column, and payments, 1.29M positions against 1.73M—so those cells compare two runs rather than two checkpoints; on the cells with their own fine-tunes the two checkpoints scored the same positions to within 1%. $*$ marks beverage discovery’s return cell, which rests on 98 positions.

Color. Blue is the numerator ahead. The scale is clipped at $10\times$ and stretched near parity, so where a cell saturates the printed number is the one to read.

F.2 Dataset Characteristics

The other tables report how each method did on each dataset. Table 7 reports how the twelve rows differ: how large each catalog is, how many positions we score on it, what event precedes the target, how much history stands in front of it, how concentrated the catalog is, how often the answer is already in the window, and what we score there zero-shot. The rows are the rows of every breadth figure, so a cell there can be read against the stream it came from.

The footer needs care, which is why the table sits here rather than in the results. It gives the rank correlation of each column with three outcomes, and twelve points is few enough that a single row moves a coefficient by around 0.2, while the columns are not independent of each other—catalog size, search share and domain travel together across these datasets, as Figure 24 sets out. Nothing in it is an effect estimate. Two near-zero coefficients are worth naming, though at twelve points they are inconclusive rather than evidence of no relationship: neither what precedes the target nor how concentrated the catalog is shows a clear rank association with our zero-shot level, at -0.02 for purchases and $+0.16$ for concentration, so the table gives no support to the two most obvious explanations without ruling them out. The association that is there runs the other way from ours: the gain from fine-tuning correlates -0.85 with the zero-shot level, so adaptation pays most exactly where transfer pays least, which is the same shape Section 3.2 reports cell by cell.

F.3 Baselines

The breadth matrices (Figures 5, 12 and 11) print one ratio per cell without saying what it is against. Table 8 describes the arms—size, target data, what each can rank, and what a ranked list costs—and Table 9 gives, on the same grid, which arms ran on each cell and which one the ratio is against: the strongest arm run there.

Position coverage. Every arm that ran on a cell is in its pool, whether or not it scored the same positions we did; some runs were cut or strided for cost. We read such scores as estimates of what the arm would get on our positions, which assumes the positions were chosen without regard to difficulty, so every cell whose winner scored a different set is marked $\sim$. Of the 29 product-grid winners, 10 scored our positions exactly, 17 fewer and 2 more. The assumption is partly testable: RecFormer was scored on two or more position sets on 38 cell-and-regime pairs, and its recalls agree to a median of 2.5%, except on full search, where a variant that selects search positions spreads by up to $1.57\times$. Admitting

	Scale		Precedes the target			In window		Conc.		Zero-shot 12.5B				
	cat. items	pos. scored	view start	search	buy	none	≥ 10	top 1k	re-peat	R@10	vs best	vs 0.5B	+ tune	
<i>Retail</i>														
Marketplace	10M	2M	8.5	11	54	0.7	95	6.4	12	23	2.0	1.2	–	
Grocery 1	30k	10M	69	31	0	0.3	97	52	3.2	41	2.9	1.2	1.2	
Grocery 2	70k	9M	56	44	0	1.0	93	41	0.7	38	2.4	1.1	1.2	
Pharmacy (browsing)	30k	4M	76	24	0	12	43	45	19	41	2.0	1.0	1.2	
Pharmacy (transactions)	40k	3M	0	0	70	30	2.9	54	<0.1	13	2.2	1.2	1.3	
Convenience	30k	200k	100	0	0	7.6	68	35	9.9	13	1.2	1.1	2.2	
Home decor	30k	800k	93	6.8	0	11	47	51	20	27	1.3	1.0	1.3	
Camera & photo	40k	200k	0	0	49	24	32	54	4.2	5.8	0.8	1.1	3.8	
<i>Engagement</i>														
Beverage discovery	20M	1M	53	46	0.2	0.9	94	7.5	5.2	37	7.1	19	1.1	
Image discovery	5M	2M	52	48	0	0.4	97	0.8	0.6	2.6	4.7	8.2	–	
Short video	90k	80k	100	0	0	20	33	–	0	0.6	2.4	4.4	2.9	
Reviews	2M	70k	100	0	0	27	22	–	0	0.8	1.6	1.2	1.6	
<i>Rank correlation of the column with:</i>														
R@10	–0.31	+0.80	–0.07	+0.59	–0.02	–0.45	+0.45	+0.16	+0.58	<i>self</i>	+0.26	–0.46	–0.85	
vs best	+0.54	+0.51	–0.15	+0.71	–0.03	–0.57	+0.57	–0.45	–0.31	+0.26	<i>self</i>	+0.62	–0.76	
+tune	–0.07	–0.78	+0.21	–0.86	+0.00	+0.60	–0.60	+0.43	–0.25	–0.85	–0.76	–0.02	<i>self</i>	

Table 7: **What the twelve datasets differ in, beside the scores.** One row per dataset, on the item slice every breadth figure is built on. *Scale*: products in the catalog, and item positions the zero-shot 12.5B was scored on, both rounded to one significant figure. *Precedes the target*: what the event in front of it was, as a percentage of those positions—a view or a session start, a full or partial query, a purchase. These need not sum to 100: marketplace and beverage discovery also log refund and unknown predecessors, 27% and 1% of their positions, and the columns are the tasks of Figure 10 rather than a partition. *In window*: items in front of the target, none of them or ten and more, which is how much of a session a method has to read on that stream; positions with none are outside every task the grids draw. *Conc.*: the share of targets inside a 1,007-item shortlist of the commonest training products, on the 10 rows that have such a run of ours, and the recall@10 of the recent-items rule, which re-offers what the session already holds. The second is a property of the stream and not of a method: it says how often the answer is already in front of the model, and it is 0 on the two streams where re-offering is never right. *Zero-shot 12.5B*: our level, then that level over the strongest arm given no target data (the envelope of Figure 24), over our own 0.5B, and the fine-tuned checkpoint over the zero-shot one. A dash is a number the benchmark does not carry: no shortlist run on the two rows scored under the protocol of Figure 1, and no fine-tune on image discovery or on marketplace. The footer gives Spearman’s ρ between each column and three outcomes, over the twelve rows or over the fewer that carry both numbers. It is a description of this table and not an effect estimate, for two reasons that both cut the same way. Twelve points is few enough that one row moves a coefficient by around 0.2, and the columns are not independent: catalog size, search share and domain travel together here—size against concentration is -0.48 over those 10 rows—so a column can correlate with an outcome through any of the others. A near-zero here is inconclusive rather than evidence of no relationship: what precedes the target and how concentrated the catalog is show no clear rank association with our level, where positions scored and search share do. The clearest reading in the table is the last row rather than any of ours: the gain from fine-tuning runs *opposite* to the zero-shot level at -0.85 , so adaptation pays most exactly where transfer pays least.

mismatched arms also costs us margin rather than buying it: on 14 of the 19 marked cells an exactly matched arm scored lower, and restricting every cell to exact matches would raise the grid’s geometric mean from $2.0\times$ to $5.3\times$.

The pool. All four learned arms—SASRec, HSTU, RecFormer and OneRec—ran on all 16 non-search cells, and every populated cell also carries a cheap rule. The three shortlist arms are in the pool wherever they ran (Appendix F.4). SASRec, HSTU, TabPFN and LightGBM are excluded from the 13 search cells, where they would rank a post-query position with the query discarded, so those comparisons fall to Gemma, RecFormer, OneRec or a rule. Gemma takes eight of them: without it, grocery 1’s two search cells and grocery 2’s partial search would stand at $21\text{--}28\times$ rather than $2.1\text{--}2.6\times$. Popularity is not target-blind: it ranks by how often an item is a target in the first 160k sessions of the dataset’s own training stream. Only recent items, user-popular and predict-last see no target data.

Sample size. Every ratio is a count divided by a count. Table 10 gives the correct predictions behind each cell, the fewest being 42 on beverage discovery’s return column, and Table 11 the positions, from 98 on beverage discovery’s return column to 9.8M on grocery 1’s view recall. Away from beverage discovery’s two rightmost cells the thinner leg never falls below 3,797 positions, so what limits a cell is the rate rather than the sample. Purchase recall on beverage discovery is the one cell with a large imbalance: 2,110 positions for us against 687 for the arm we are measured on.

Payments. Eleven arms were scored on next merchant and next category, at hit@5. Two session heuristics lead the pool, repeating what the session has carried (0.623 and 0.469) and answering from the user’s own history (0.610 and 0.432): both attributes are heavily repeat-driven. TabPFN v3 and LightGBM predict the label directly (0.497 and 0.279; 0.376 and 0.051), against a rule that answers the five commonest labels (0.447 and 0.137). RecFormer, OneRec and Gemma reach the label as we do, by ranking products and reading it off them; SASRec and HSTU sit at the floor on both tasks. Against the strongest arm we lead by $1.18\times$ and $1.30\times$, and at hit@1 the result is the same, 0.630 against 0.533 on merchant and 0.359 against 0.300 on category. Every arm in this block, ours included, is reported as a table of hit@k with no position count, so its evaluation windows cannot be checked against each other.

Serving cost in Table 3. An ID-based sequential recommender and ours both encode the history once and score candidates by one matrix product, so both cost a single pass. A count-based rule needs no pass at all, and is the floor the latency column should be read against. A text-encoder retriever also ranks the open catalog in one pass, but through a language encoder over the history rendered as text. A per-candidate scorer applies its model once per shortlist item, n applications for a shortlist of n , and cannot leave that shortlist. A semantic-ID generative recommender decodes each item as a code of m levels under a beam of width b , costing mb passes, or 40 at the $m=4, b=10$ typical of these systems; the beam explores only a fraction of the code space, so its list is drawn from the catalog only approximately. A prompted LLM decodes T tokens autoregressively, over a far larger model, after a prefill over the candidates in its context, and ranks only the shortlist it is given, so its row is timed on a strictly smaller ranking problem than the others.

Latencies are read from each run’s recorded `ms_per_query`, taking the representative arms of each route (the rules, SASRec with HSTU, and RecFormer’s two arms pooled; TabPFN and LightGBM; OneRec; Gemma 4 31B; our zero-shot 12.5B; Table 8 gives every arm). They are harness throughput figures, not serving latency: the harness batches, the runs span four hosts, our arm runs hand-optimized kernels where the baselines run as research code, and within one arm at a fixed catalog the number moves by tens, so the column is read down, for the two orders of magnitude separating a single-pass ranker from a decoding LLM, and not across a row. The passes column is the implementation-independent half of the claim and carries the argument; a serving measurement proper (percentile latency and cost per request on one harness at a stated catalog size) is left to future work.

Latency in Figure 3. Our zero-shot and fine-tuned checkpoints of one size share one latency, the median over datasets of their runs on B200 hosts. Two zero-shot runs scored on a slower host, and the fine-tunes’ absence from the two largest catalogs, would otherwise separate them for no reason in the model.

F.4 Large Tabular Models and an LLM Ranker

The tabular models treat each product as a class and cannot rank a catalog of millions; TabPFN in particular is built for tabular classification over a modest number of classes, not for choosing among a large catalog. An LLM ranks only the candidates in its context window (Appendix F.5): listing a catalog of tens of thousands to millions of products in every prompt is prohibitively expensive at best and beyond any context window at worst. TabPFN v3, LightGBM and Gemma 4 31B therefore each rank a shortlist of the most frequent products in the target’s own data: 1,000 on ten rows, and 167 on reviews and for TabPFN on short video. None of the three is therefore target-blind, their runs without samples included: the shortlist carries the dataset’s own item frequencies, which our zero-shot model sees only as each item’s own past interaction count up to the scored position (Appendix B.1), not as a list of the most frequent items. A target outside the shortlist is scored as a miss, just as an item we fail to retrieve is, so both sides measure the same quantity: the share of positions whose target lands in the top ten. That is why the three sit in the pool of both breadth matrices. Gemma is the strongest baseline on nine of the 31 cells, eight of them search cells, and the tabular pair on none. This appendix compares us against the three alone.

Against LTM and the LLM ranker. Figure 22 divides our zero-shot full-catalog recall by the best of the three. We lead 29 of 30 cells by a geometric mean of $4.5\times$. The one loss is camera & photo’s purchase recall, at $0.90\times$ to Gemma. Gemma is the strongest of the three on every cell. It alone contests the search columns: the tabular arms see only brand, category and event type, never the query, and are barred there as SASRec and HSTU are. Nearly all of the margin comes from the shortlist, not from ranking skill. Credit the three with perfect reachability at the skill they show on their shortlist and the aggregate inverts, to $0.73\times$, with them ahead on 22 of the 28 cells. That counterfactual assumes ranking fourteen million items is as easy as ranking a thousand, so it is not an alternative reading. It shows that an LLM ranks a shortlist well but cannot produce one that holds the answer.

Caveats. On ten rows the three were scored on the first 2,000 evaluation sessions, 0.3–33% of our positions; on short video and reviews they score every position ours do. Reachability is 35–55% on the seven smaller catalogs but 8% or less on marketplace, beverage discovery, reviews, image discovery and short video, which carry the largest cells.

Arm	Size	0-shot	Target data	Ranks	Item as	Hist.	Passes	ms/query
<i>Ours</i>								
BehaviorGPT-v4, zero-shot	12.5B	Yes	none	catalog	multimodal	512 ev.	1	0.71 (0.06–2.4)
LoRA fine-tuned	12.5B	No	LoRA $r=128$	catalog	multimodal	512 ev.	1	0.47 (0.16–2.2)
BehaviorGPT-v3, zero-shot	0.5B	Yes	none	catalog	multimodal	512 ev.	1	0.22 (0.01–2.3)
fine-tuned	0.5B	No	all weights	catalog	multimodal	512 ev.	1	0.20 (0.10–2.2)
<i>Trained on the target’s training split</i>								
SASRec [62]	— ^a	No	from scratch	train items ^b	item IDs	512 ev.	1	0.18 (0.02–1.6)
HSTU [112]	— ^a	No	from scratch	train items ^b	item IDs	512 ev.	1	0.26 (0.02–1.5)
RecFormer fine-tuned [72]	0.11B ^c	No	all weights	catalog	item text	512 ev.	1	5.6 (0.94–19)
<i>Given no target interactions</i>								
RecFormer zero-shot [72]	0.11B ^c	Yes	none	catalog	item text	512 ev.	1	5.8 (2.3–19)
OneRec 8B Pro [115]	8B	Yes ^d	none	code space ^e	item text	512 ev.	mb	20 (0.36–38)
<i>Cheap rules</i>								
popularity	none	No	target counts ^f	catalog	item IDs	none	0	0.04 (0.02–1.4)
recent items	none	Yes	none	session window	item IDs	20 ev.	0	0.05 (0.02–1.4)
user-popular	none	Yes	none	session window	item IDs	20 ev.	0	0.04 (0.02–1.4)
predict-last	none	Yes	none	session window	item IDs	1 ev.	0	0.04 (0.02–1.3)
most-frequent label ^g	none	No	gold counts	merch., cat.	labels	none	0	—
<i>Bounded to a candidate list</i>								
TabPFN v3 [49]	— ^h	No	50k rows ⁱ	1,007 shortlist	brand, cat.	20 ev.	56 ^j	3.7 (0.94–57)
LightGBM	300 rounds ^k	No	50k rows ⁱ	1,007 shortlist	brand, cat.	20 ev.	trees ^k	0.59 (0.41–0.84)
Gemma 4 31B	31B ^l	Yes	none	1,007 shortlist	item text	20 ev. ^m	prefill+ T	34 (3.6–186)

Table 8: **What each arm in the benchmark is, and what a ranked list costs it.** Where each arm ran is in Table 9, its accuracy in Table 14. *Size*: non-embedding parameters, as in Table 1. *0-shot*: no interactions from the target dataset. *Target data*: what the arm was given; trained and fine-tuned arms use the target’s training split. *Ranks*: the set the returned list is drawn from; a target outside it counts as a miss. *Item as*: how a candidate reaches the arm, and so what can carry it to an unseen catalog. *Hist.*: what the arm reads from the session; the harness gives every sequence model 512 events, and shorter figures are the arms’ own. *Passes*: sequential forward passes per request. Ours, the specialists and RecFormer encode the history once and score a cached catalog by one matrix product; OneRec decodes m code levels under a beam of b ; an LLM prefills its candidates and decodes T tokens (Section 2.8). *ms/query*: median over the arm’s datasets, with the range. It is an order of magnitude, not a benchmark: the harness batches, runs span four hosts, and one arm varies widely (Gemma 3.6–186 ms on the same 1,007 candidates; ours 0.06–2.4 ms as its catalog grows from 27,064 to 18.4M items). The last block ranks only 1,007 candidates, which flatters its timings.

^aTrained per dataset by us; the result package records no architecture, so no count is given. Both are dominated by an item table with one row per catalog item, which is the quantity this column excludes. ^bOnly items appearing in the target’s training interactions, 8.6% of the catalog on reviews, a target outside them counting as a miss. ^cA Longformer-base backbone; item text vectors are precomputed, so the pass is over the history alone. ^dZero-shot in our runs, which is what the package records and what Figure 1 reports: it reaches an unseen catalog by decoding codes rather than by being trained on the target, which is why it sits in this block and not the one above. ^eA semantic-ID decoder reaches an unseen catalog by decoding codes rather than looking up rows, but a beam explores only part of the code space, so the reachable set is not the whole catalog. ^fHow often an item is a target in the first 160k sessions of the dataset’s own training stream. It reads nothing from the session in front of it and is the only rule here given target data. ^gPayments only, where the two attribute tasks are scored at hit@5: the rule answers the commonest merchant or category labels, computed from the counts the run itself records. ^hNot published for v3. TabPFN v2 is 0.026B. ⁱFifty stratified rows per class over the 1,000-class candidate list, drawn from the target’s training stream: in context for TabPFN, fitted for LightGBM. A row is brand, first category and event type per history slot, carrying no product identifier, text or imagery. ^jFourteen ECOC sub-classifiers times four ensemble members, each re-encoding the support set, which is why a 1,007-item ranking costs more than a full-catalog pass of ours. ^kThree hundred boosting rounds at 63 leaves over 1,000 classes, so some 3×10^5 trees, ranked on pre-softmax margins. ^lAs named by the model, total rather than non-embedding parameters and not our count. ^mIts prompt renders the user’s 20 most recent interactions and then the numbered candidate list, in text, and asks for a JSON array of the top- n candidate numbers under greedy decoding (Section 2.9; Appendix F.5).

LightGBM barely learns: within its shortlist it reaches 0.9–2.3% recall@10 on the ten benchmark rows, against about 1% for a random ordering of the list. On grocery 1 it fits its training positions at 7.8% recall@10 but reaches only 1.3% on held-out ones; regularizing it with fewer boosting rounds narrows the gap, to 5.9% and 1.8%, without lifting it far from chance, and the tables keep the configured fit. A gradient-boosted classifier over a thousand item classes, with only brand, category and event type per history slot as features and no item identity, text or query, is hard to configure so that it generalizes on catalogs of this size. Tables 27–29 give all three arms cell by cell.

The catalog bound, measured. Figure 23 gives every arm two measured points: recall on a 1,000-item catalog, over the positions its shortlist reaches, and recall on the dataset’s own catalog. Our recall falls by a geometric mean of $0.44\times$ between the two and theirs by $0.21\text{--}0.23\times$, so our margin over Gemma roughly doubles, from $1.75\times$ to $3.52\times$. The slopes differ in kind. Ours is the cost of choosing among more items. Theirs is reachability alone: they rank the same thousand items equally well at both ends, but the target usually falls outside them. Below 70,000 items the lines run roughly parallel. The two largest catalogs separate them: on beverage discovery we fall from 70.1% to 36.5% where Gemma falls from 48.3% to 4.1%. Camera & photo, the most concentrated row, with 54.5% of targets inside a thousand items, is the one row where Gemma leads at both ends: a list of its thousand most frequent products, counted from its own data, reaches more than half its targets, a prior our zero-shot model is not given as a list.

	View recall	Full search	Partial search	Purchase recall	Item returned
<i>Retail</i>					
Marketplace	r ~ SHRZOpulTBG	R ~ ZOprulG	r ~ RZOprulG	R ~ SHZOprulTBG	r ~ SHRZOprulTBG
Grocery 1	S HRZOprulTBG	G ~ RZOprul	G ~ RZOprul	.	.
Grocery 2	S HRZOprulTBG	G ~ RZOprul	G ~ RZOprul	.	.
Pharmacy (browsing)	R ~ SHZOprulTBG	G ~ RZOprul	G ~ RZOprul	.	.
Pharmacy (transactions)	.	.	.	H SRZOprulTBG	.
Convenience	H SRZOprulTBG	.	.	.	.
Home decor	S HRZOprulTBG	R ZOprulG	R ZOprulG	.	.
Camera & photo	.	.	.	S HRZOprulTBG	.
<i>Engagement</i>					
Beverage discovery	r ~ SHRZOpulTBG	G ~ RZOprul	G ~ RZOprul	r ~ SHRZOpulTBG	r ~ SHRZOpulTBG
Image discovery	u ~ SHRZOprulTBG	r ~ RZOprulG	.	.	.
Short video	R SHZOprulTBG	.	.	.	.
Reviews	G SHRZOprulTB	.	.	.	.
<i>Payments</i>					
	Next merchant		Next category		
Cross-merchant	r [†] u [†] R [†] T [†] Z [†] p [†] f O [†] G [†] B [†] H [†] S [†]		r [†] u [†] G [†] R [†] T [†] Z [†] O [†] p [†] f S [†] H [†] B [†]		

Table 9: **Which baselines were run on each dataset and task, and which one Figures 5, 12 and 11 are measured against.** The grid is the one in those figures. The leading, **bold** code in each cell is the arm the ratio is against—the highest recall@10 of any arm run on that cell—and the codes after it are the rest of the pool. Trained on the target’s training split: **S** SASRec, **H** HSTU, **R** RecFormer. Counted over its training stream: **p** popularity. Given no target data: **r** recent items, **u** user-popular, **I** predict-last, which rank from the eval session alone, and **Z** RecFormer zero-shot, **O** OneRec 8B Pro. An arm’s code is black if it scored exactly our evaluation positions, **blue** if it scored fewer and **orange** if it scored more; a cell whose winner is blue or orange carries a ~ here and in the figures. **Green** arms rank only a candidate list of 1,000 products (167 on the last two rows), a target outside it counting as a miss: **T** TabPFN v3, **B** LightGBM, **G** Gemma 4 31B. **S**, **H**, **T** and **B** are query-blind and so absent from the search columns. In payments, [†] marks an arm reported as a table of hit@k with no position count, **f** is a most-frequent-label rule, and the block is scored at hit@5. A . is a task the dataset does not log. Across the 31 populated cells the winner is Gemma 4 31B on 9, recent items on 9, RecFormer on 6, SASRec on 4, HSTU on 2, user-popular on 1. The text of this appendix discusses what the pool is worth.

Across datasets. Figure 24 asks whether the margin is also larger on the larger assortments. It correlates with catalog size for every arm, Spearman +0.34 to +0.73, but the benchmark cannot support a causal reading. Catalog size tracks concentration ($\rho = -0.48$) and domain: three of the four catalogs over a million products are engagement streams, and seven of the eight retail catalogs lie between 27,000 and 67,000 products. Marketplace, the one large retail catalog, has a margin of $2.0\times$ over the strongest zero-shot arm, inside the small retail rows’ range of $0.5\text{--}2.9\times$. Against Gemma alone it is $10.6\times$, but only because a session-repeat rule beats Gemma more than fivefold on that stream. The three session rules have no panel of their own because their ratio can be undefined: *predict-last* scores zero on seven of the ten datasets it ran on. Normalizing the axis would not help, since any per-dataset factor cancels out of a ratio.

F.5 LLM Ranking Protocol

This subsection carries the full protocol behind the LLM arm of Section 2.9. The candidate list and task instructions form a constant, domain-neutral system prompt, and only the history varies per request: the user’s 20 most recent interactions, oldest first, each rendered as its catalog text—name, brand, categories—in the same format as the candidates. The model is instructed to return a JSON array of the top- n candidate numbers, n being the largest k we report plus the number of session tokens, with greedy decoding (temperature 0) and its thinking mode disabled. A malformed response receives one format-repair round trip; a response that still fails counts as a non-prediction, scored as uniform over the candidates. Returned ranks are mapped to scores with a Zipf weighting, so the ranking metrics (Recall@ k , NDCG, MRR) reflect the model’s stated ranking exactly; we report no likelihood-based metrics for this arm. Session-boundary tokens are removed from the ranking and metrics are product-only. In the N -shot variant, N (history, next-item) pairs from the training split are added to the system prompt and no parameters are updated.

F.6 Decision Models as Shortlist Rankers

Jev [1] is TypeSafe AI’s “System One” decision model, proprietary and served only over an API: given a state and a typed question whose options are fixed in advance, it returns a calibrated probability over those options in one parallel pass and generates no text. Two open-weight models of that shape rank the shortlist here under the protocol above: Von 1.2 [86], 0.4B, a ModernBERT-large encoder with one option-marker head, and Kev [85], 9B, a LoRA adapter and pointer head on a hybrid Qwen3.5 backbone. Both are zero-shot and rank the same 1,000-product shortlist Gemma

	View recall	Full search	Partial search	Purchase recall	Item returned
<i>Retail</i>					
Marketplace	43k/30k	18k/4861	35k/6590	167k/31k	105k/103k
Grocery 1	2370k/1278k	1848k/900k	1550k/593k	.	.
Grocery 2	1005k/711k	1475k/750k	1155k/505k	.	.
Pharmacy (browsing)	1037k/945k	336k/206k	184k/113k	.	.
Pharmacy (transactions)	.	.	.	378k/269k	.
Convenience	23k/23k	.	.	.	.
Home decor	201k/174k	9873/6122	6694/4229	.	.
Camera & photo	.	.	.	7639/17k	.
<i>Engagement</i>					
Beverage discovery	75k/49k	73k/11k	254k/25k	685/528	82/41
Image discovery	30k/5403	22k/5816	.	.	.
Short video	472/347	.	.	.	.
Reviews	532/324	.	.	.	.
<i>Payments</i>					
	Merchant		Category		
Cross-merchant	1276k/1080k		1055k/813k		

Table 10: **How many correct predictions each cell of Figures 5, 12 and 11 rests on.** Same grid as Table 9. Each cell gives the zero-shot 12.5B’s recall@10 hits and then those of the arm its ratio is against—the **bold** code in the cell above—over the same positions, so the two are the numerator and denominator of the printed ratio up to a common count. Hits rather than positions, which Table 11 gives on the same grid: away from beverage discovery’s two rightmost cells positions are never the scarce quantity, and what a cell can be short of is successes to count. No cell in the grid has a leg under 30 hits: the thinnest is beverage discovery’s return column at 42, and the next is reviews’ view recall at 324, so every ratio here is readable as a factor. A count of h carries a relative standard error near $1/\sqrt{h}$, so at 30 hits a leg is good to about 18% and at 4 to about 50%; below that a ratio is a comparison of small integers and should be read as a direction and not a factor. The table is drawn from the zero-shot run because that is Figure 11’s arm; the fine-tuned run scores at least as many hits on every cell it shares, or clears 30 where it does not, so a cell thick enough here is thick enough in Figures 5 and 12. Payments is hit@5 on an attribute rather than recall@10 on an item, and its counts are large enough that the distinction does not matter here.

does (Appendix F.4), each applied as released under one prompt, and neither was trained on behavioral data. What follows bounds what a general decision model does on this task without adaptation, not what one trained for it would. Neither arm is in the pool of either breadth matrix, so no cell in the paper rests on them.

Table 12 reads recall over the positions the shortlist reaches—the target inside the list—and not over every scored position as the matrices do, so all four arms are being measured at the ranking step alone, with the reachability bound set aside. That is why the numbers here run far above the same arms’ cells in Figure 22.

Results. Neither open decision model outranks the LLM on any row. On Grocery 2 Kev reaches $0.74\times$ Gemma’s recall@10 and Von $0.71\times$, both $3.5\text{--}3.6\times$ shortlist popularity. On marketplace Kev matches shortlist popularity at recall@10 and leads it at recall@100, 48.0% against 39.9%, while Von falls below both. On short video Kev sits level with Gemma, 3.1% against 3.2%, but every text arm there is within a factor of four of chance, which is 1% on a thousand-item list.

They rank on the last event. Two fifths of Grocery 2’s product positions follow a search, and there the three reach 67.3% (Gemma), 51.4% (Kev) and 43.9% (Von) by matching the query against item text. Those three are over the search positions all three arms scored, so they are lower than Gemma’s own run read alone, which pools to 68.3% across the two search columns. Von reads only the latest event, by design: given the full history, as Gemma and Kev are, it scores 3.1% on a 5,000-position Grocery 2 sample against 15.9% on the latest event alone. It was trained to judge short, single-situation premises and scores how well an option fits the premise, so a longer history rewards candidates related to any past item rather than to the next one.

Against our own checkpoint. Held to the same shortlist and the same reachable positions, our zero-shot 12.5B reaches 54.2% at recall@10 on Grocery 2 and 80.9% on marketplace, which is $2.0\times$ and $4.8\times$ Kev, the stronger of the two, and $2.1\times$ and $28.9\times$ Von. That is the comparison this table supports and the narrowest one available, both sides ranking a thousand items. It is not the task the paper measures, and the difference matters in both directions: the shortlist is a thousand frequent products of the target’s own stream, so it flatters every arm that gets it—on marketplace Gemma’s 26.8% here is above our 23.4% over the full catalog, where the target can be any of ten million items. Figure 22 accounts for that bound, and Gemma is the denominator on every one of its cells, so Kev and Von, below Gemma on all three rows here, would sit further behind still.

	View recall	Full search	Partial search	Purchase recall	Item returned
<i>Retail</i>					
Marketplace	147k/ 111k	51k/ 69k	138k/ 102k	932k/ 535k	127k/ 98k
Grocery 1	9.8M/9.8M	2.2M/ 7.7k	2.1M/ 8.7k	.	.
Grocery 2	5.3M/5.3M	1.9M/ 20k	2.3M/ 25k	.	.
Pharmacy (browsing)	2.9M/ 1.5M	484k/ 6.7k	426k/ 5.9k	.	.
Pharmacy (transactions)	.	.	.	2.3M/2.3M	.
Convenience	184k/184k	.	.	.	.
Home decor	753k/753k	28k/28k	27k/27k	.	.
Camera & photo	.	.	.	78k/78k	.
<i>Engagement</i>					
Beverage discovery	586k/ 233k	84k/ 3.8k	428k/ 20k	2.1k/687*	98/26*
Image discovery	1.0M/ 1.2M	960k/ 825k	.	.	.
Short video	79k/79k	.	.	.	.
Reviews	70k/70k	.	.	.	.
<i>Payments</i>					
	Next merchant		Next category		
Cross-merchant	1.7M/1.7M		1.7M/1.7M		

Table 11: **How many evaluation positions each cell rests on.** Same grid as Table 9. Each cell gives our positions and then those of the arm the ratio is against—the **bold** code in the cell above. Unlike the hit counts of Table 10 these are a property of the benchmark and not of any run: they do not move when a model improves, so this is the table to read for sample size. One table serves all four matrices, our own checkpoints scoring the same positions on every cell they share apart from marketplace, where the 0.5B scored a narrower window and the scale figures print a +. Our counts run from 98 to 9.8M, and away from the two cells marked * the thinner leg never falls below 3,797, so what limits a cell is almost always the rate and not the sample. **Red** and * mark the two cells where either leg falls under 1,000 positions: beverage discovery item returned (98 against 26), which the figures star for the same reason, and beverage discovery purchase recall (2,110 against 687), which they do not, their * counting our positions alone where here it is the baseline that was cut. Both read as a direction rather than a factor. The second leg carries Table 9’s ink: **blue** where the winning arm scored fewer of our positions, **orange** where it scored more, in both cases because runs were cut or strided for cost and not because the protocol differed. The spread is wide—grocery 1 full search admits an arm scored on 0.4% of our positions and marketplace full search one scored on $1.35\times$ them—and it is what every $\sim$ in the figures rests on: that a cut or widened position set was chosen without regard to difficulty. Payments is one stream scored once and its baseline is a rule over the same positions, so its two legs are equal.

F.7 What Adaptation Adds

Figures 5 and 11 are the same pretrained model with the target’s data and without it, each measured against the field, so the difference between them carries the gain adaptation brings together with how strong the field happens to be on a row. Figure 25 takes the field out: it divides each fine-tuned model by its own zero-shot base, on the same positions, so a cell is what the dataset’s own training split was worth with pretraining held fixed. It is the quantity a practitioner with a stream and a labelling budget is deciding about, and the one the two grids above can only imply.

Adapting both stages. The fine-tunes in these grids adapt stage 1 and then stage 2. Adapting stage 2 alone, on the same rows and the same positions, recovers most but not all of the gain (Table 13): both stages lead on 19 of the 20 cells, by $1.10\times$, and the one tie is beverage discovery’s return cell on 98 positions. The difference is largest on view recall, $1.20\times$, and smallest on the two search columns, $1.05\times$ and $1.06\times$. Over the zero-shot checkpoint, stage 2 alone gains $1.17\times$ and regresses on beverage discovery’s view and purchase cells; adapting both stages gains $1.29\times$ and improves every cell.

F.8 Per-Method Accuracy on Every Cell

The breadth matrices print ratios. Tables 14–26 print the accuracies behind them, one table per method on the same grid, and Tables 27–29 do the same for TabPFN v3, LightGBM and Gemma 4 31B. Every ratio in the breadth results can be recovered from them. A cell of Figure 11 is our 12.5B zero-shot entry divided by the highest entry of any other method on that cell, the three candidate-bounded tables included. Figures 2 and 10 summarise the same numbers.

Conventions. Values are recall@10 in percent, and hit@5 in the payments block. **Bold** marks the leader of a cell over all sixteen tables, so each cell is bold in exactly one. Our checkpoints hold every bold: the fine-tuned 12.5B on 21 of the 29 grid cells, the zero-shot 12.5B on 7 and the fine-tuned 0.5B on 1. RecFormer’s run variants collapse to its strongest on each cell, as in the figures’ pool; this only lowers our margins. Repeated seeds are averaged. **Blue** and **orange** mark runs scored on fewer or more of our positions, and a bold on a mismatched count carries a $\sim$. Query-blind arms are

BehaviorGPT 12.5B zero-shot over the best of TabPFN v3, LightGBM and Gemma 4 31B, each ranking the whole catalog it can reach

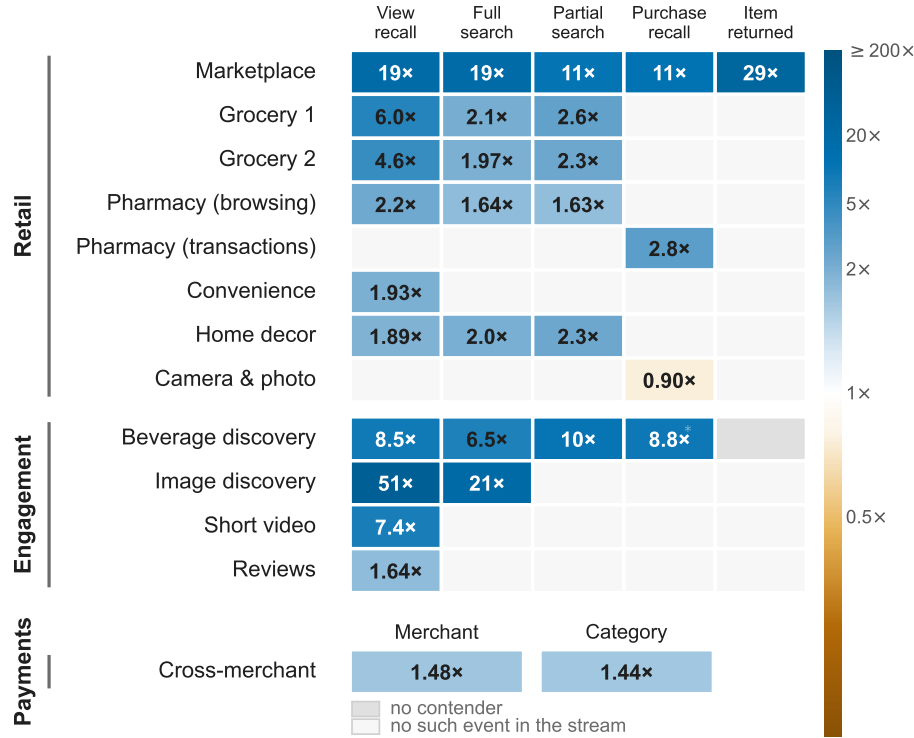


Figure 22: **A model that can only consider a thousand items is not doing the task, and the score says so.** Zero-shot 12.5B recall@10 over the whole catalog, divided by the best of TabPFN v3, LightGBM and Gemma 4 31B on their shortlist of frequent training products (1,000 items; 167 on reviews and for TabPFN on short video), a target outside the list counting as a miss. We lead 29 of 30 cells, geometric mean 4.5×; the color scale runs to 200× rather than 10×. Gemma is the denominator on every cell, and alone on the search columns. Beverage discovery’s return column is grey because none of the three ranks any of its 18 positions inside ten. Payments is hit@5 against the best of the three over the same stream, TabPFN on next merchant and Gemma on next category; there the tabular pair rank the labels directly and in full, so no shortlist bound applies. On ten rows their runs cover the first 2,000 sessions only. *Fewer than 1,000 positions in their run.

greyed in the search columns and barred from the bold there. A dash is a run that does not exist; a · is a task the dataset does not log. The candidate-bounded tables rank a 1,000-item shortlist, 167 on two rows, over the first 2,000 evaluation sessions except on short video and reviews (Appendix F.4). In payments every cell is read from the hit@k table behind Table 9, and the tabular pair rank the labels directly rather than a product shortlist.

Reading Figure 2. Each cell is a method’s recall as a share of the best on that dataset. Rows are sorted by mean rank among the methods run on each dataset and payments task. Rank tolerates the exact zeros some rules post, where a geometric mean would be undefined, and the median column beside it gives magnitude. *Predict-last* is dropped, being zero on seven of its ten datasets. Marketplace and image discovery have no fine-tune, so each fine-tuned row carries its zero-shot cell there. The **B** arms rank a shortlist on a subsample of positions, so their cells are not on quite the same footing as the rest. HSTU, RecFormer, SASRec and Gemma sit within a rank of each other, and their order depends on the summary statistic. Gemma’s place in that cluster comes from search: on view recall alone it is eleventh, at 14.0% of the leader. The payments band counts toward the rank and the median as two columns, scored by hit@5, with each fine-tuned row carrying its zero-shot cell there as no payments fine-tune exists. Its two tasks share one event stream, so payments weighs two columns against each catalog’s one; scoring it as one column instead would move no row of ours relative to any baseline.

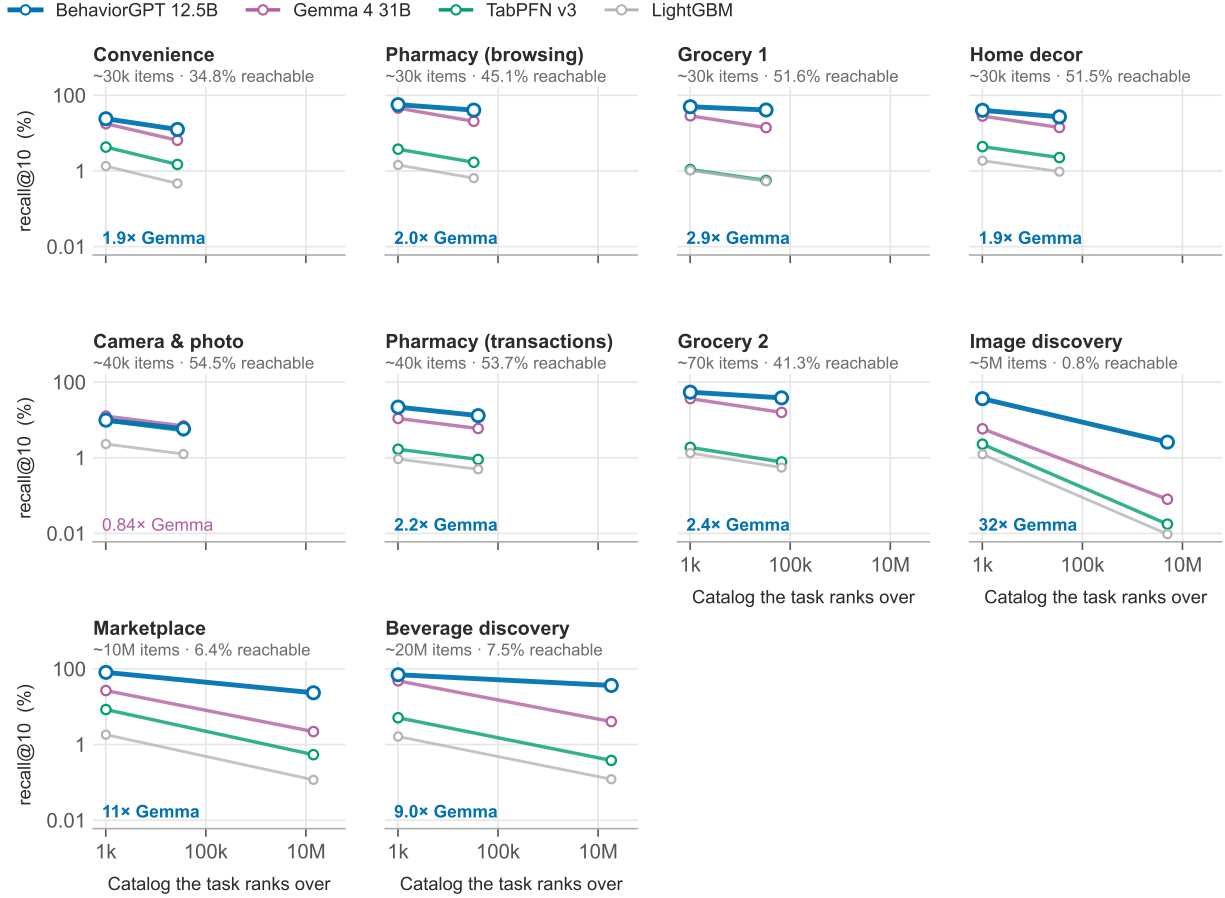


Figure 23: **What a larger catalog costs an arm that can rank it, and one that cannot.** Zero-shot recall@10 on the product slice, two measured points per arm per dataset. *Left*, at 1,007 items: recall over positions whose target is inside the arm’s candidate list. *Right*, at the dataset’s catalog: recall over every position, which for the bounded arms is the left point times reachability. Panel labels give our recall over Gemma’s at the right-hand end: 1.9 to 32× on the nine rows we lead, 3.52× over all ten, against 1.75× at the left. Camera & photo is the one row Gemma leads, at 0.84×. Shortlist runs cover the first 2,000 sessions. Short video and reviews are absent, having no shortlist run of ours.

Reading Figure 10. Split by task, four different arms are the strongest baseline across the seven columns. The session rules take item returned, recent items and user-popular at 71% and 63% of our level, since a returned item is usually one the session already holds. They also take both payments columns, where merchants and categories repeat more readily than products. HSTU leads view recall at 34% and SASRec purchase recall at 28%. Gemma takes both search columns at 37% and 35%, about three times the next arm, because it reads the query. TabPFN reaches 68% and 46% on the payments columns, the attribute tasks it was built for, against 1.4–3.4% on the product tasks.

zero-shot 12.5B over each arm given no target data, twelve datasets apiece · *TabPFN is fitted on target rows, drawn for reference last panel: the strongest of six such arms, three being session rules, TabPFN not among them

● retail ● engagement

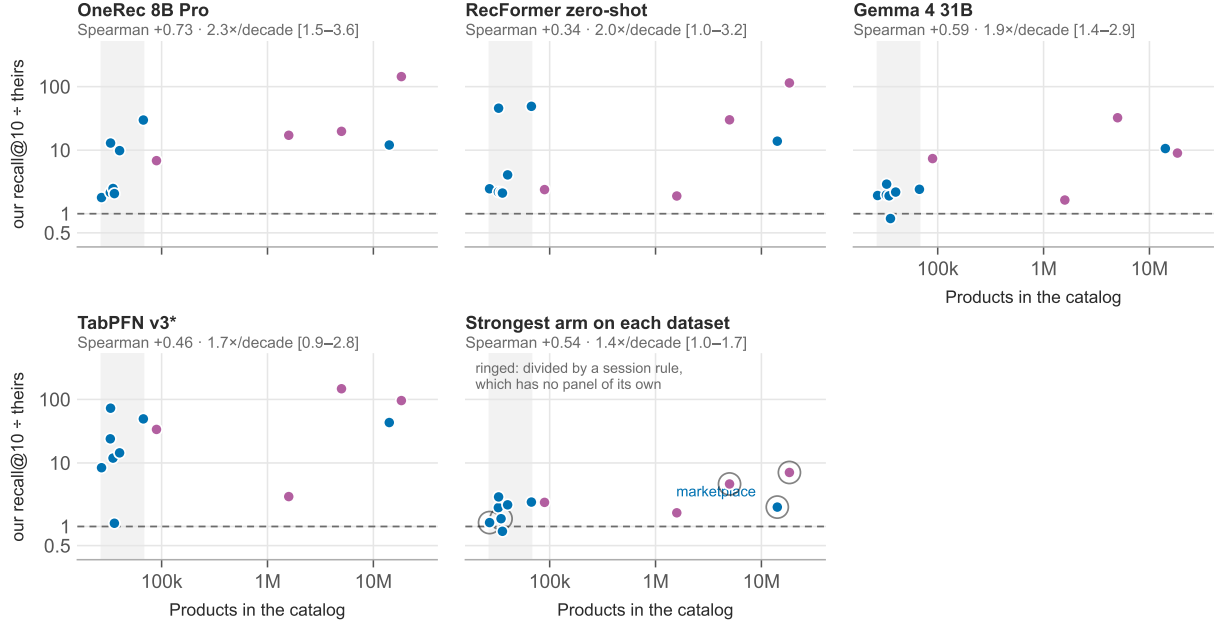


Figure 24: **The margin rises with catalog size, and with everything catalog size travels with here.** Zero-shot 12.5B recall@10 on the product slice over each arm given no target data, against catalog size, both axes logarithmic, the dashed rule parity. LightGBM and *popularity* are excluded, both being fitted or counted on the target’s training split. *TabPFN v3 is too, and is drawn for reference only, as the most prominent large tabular model; it is not in the last panel’s pool. The shaded band, 27,064 to 67,209 products, holds seven of the eight retail catalogs. The last panel divides by the strongest of the six such arms on each dataset, three of them session rules, ringed where one is the denominator. Each panel prints Spearman’s ρ and a log-log slope per tenfold increase in catalog, with a bootstrap interval. No line is drawn: R^2 is 0.17–0.52, and dropping image and beverage discovery takes the slopes from $1.37\text{--}2.27\times$ to $1.06\text{--}1.70\times$ per decade, with R^2 at most 0.32.

Dataset	Arm	R@1	R@10	R@100
Grocery 2	Ours, 12.5B zero-shot	24.9~	54.2~	80.2~
	Gemma 4 31B	15.1	36.6	50.0
	Kev (9B)	7.6	27.1	47.5
	Von (0.4B)	6.3	26.1	49.8
	shortlist popularity	1.3	7.5	29.1
Marketplace	Ours, 12.5B zero-shot	29.7~	80.9~	97.8~
	Gemma 4 31B	11.3	26.8	51.8
	Kev (9B)	3.4	16.9	48.0
	Von (0.4B)	0.6	2.8	27.2
	shortlist popularity	3.1	16.2	39.9
Short video	Ours, 12.5B zero-shot	-	-	-
	Gemma 4 31B	0.5	3.2	15.6
	Kev (9B)	0.5	3.1	15.3
	Von (0.4B)	0.1	1.0	11.1
	shortlist popularity	0.1	0.9	9.9

Table 12: **Two open decision models against the LLM ranker and against us, all ranking the same shortlists.** Recall in percent over the positions the shortlist reaches, so a target outside the list is excluded here rather than counted a miss as it is everywhere else in this appendix. Our row is our own shortlist run, not the full-catalog run the matrices draw, and it carries a ~ because it covers more of the stream than the other three: 79,404 reachable positions against Gemma’s 43,310 on Grocery 2 and 17,679 against 7,501 on marketplace. We have no shortlist run on short video. *Shortlist popularity* ranks the same list by target counts in the training stream, and is not the full-catalog *popularity* arm of the matrices.

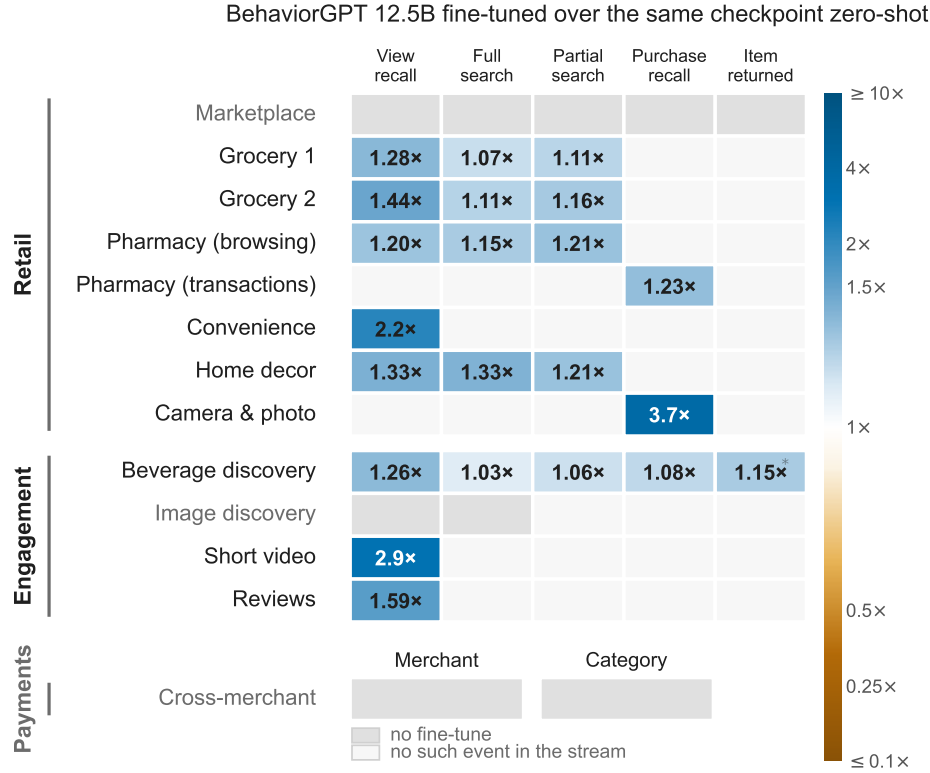


Figure 25: **Target data buys most where the zero-shot checkpoint has least to go on, and almost nothing where it is already strong.** The fine-tuned 12.5B over the zero-shot 12.5B, same rows, columns and positions as Figure 5, with the field out of the ratio. Adaptation gains on all 22 cells that have a fine-tune, by a geometric mean of $1.35\times$, and the gain is graded rather than uniform across the tasks. The two search columns move least, $1.13\times$ and $1.15\times$, and beverage discovery’s two are the smallest gains on the grid, $1.03\times$ and $1.06\times$. A checkpoint that can already read a query has the position’s answer largely in front of it, so the target’s sessions add little. A view or a purchase to condition on is worth $1.57\times$ and $1.70\times$, and the return column $1.15\times$. Nothing is carried in this figure. A row with no fine-tune is grey rather than a ratio, because carrying the zero-shot run into the numerator would divide a run by itself and print a gain of $1.00\times$ where nothing was measured: marketplace’s five cells, image discovery’s two and both payments cells, the nine the other grids mark z . *Beverage discovery’s return cell rests on 98 positions.

	View recall	Full search	Partial search	Purchase recall	Item returned
Grocery 1	25.5 / 31.0	87.6 / 89.9	76.2 / 79.9	-	-
Grocery 2	21.7 / 27.1	83.9 / 86.5	56.4 / 59.7	-	-
Pharmacy (browsing)	38.9 / 42.5	77.3 / 80.0	50.2 / 52.2	-	-
Pharmacy (transactions)	-	-	-	17.1 / 20.1	-
Convenience	23.1 / 26.9	-	-	-	-
Home decor	30.2 / 35.6	40.8 / 47.4	26.7 / 29.6	-	-
Camera & photo	-	-	-	35.0 / 35.8	-
Beverage discovery	12.3 / 16.2	88.2 / 89.6	59.6 / 63.1	31.9 / 35.2	95.9 / 95.9
Both over stage 2	1.20×	1.05×	1.06×	1.10×	1.00×

Table 13: **Adapting both stages beats adapting stage 2 alone.** Recall@10 in percent of the 12.5B fine-tuned on stage 2 alone and then on both stages, stage 1 followed by stage 2, on the rows and positions of Figure 5; the better of the two is in **bold**. The last row is the geometric mean of the second over the first down each column. Short video and reviews have no both-stage run on their split and are left out, as are the rows with no fine-tune. Over all 20 cells both stages lead by $1.10\times$.

	View recall	Full search	Partial search	Purchase recall	Item returned
<i>Retail</i>					
Marketplace	29.2	35.9	25.4	17.9	82.9
Grocery 1	24.3	84.2	72.2	-	-
Grocery 2	18.8	77.9	51.3	-	-
Pharmacy (browsing)	35.5	69.6	43.2	-	-
Pharmacy (transactions)	-	-	-	16.3	-
Convenience	12.5	-	-	-	-
Home decor	26.7	35.7	24.5	-	-
Camera & photo	-	-	-	9.79	-
<i>Engagement</i>					
Beverage discovery	12.9	87.2	59.4	32.5	83.7
Image discovery	2.92	2.24	-	-	-
Short video	0.594	-	-	-	-
Reviews	0.763	-	-	-	-
<i>Payments</i>					
	Merchant		Category		
Cross-merchant	73.6		60.9		

Table 14: **BehaviorGPT 12.5B zero-shot**: recall@10 as a percentage on every cell of the grid, hit@5 in the payments block. **Bold** where it leads every other method in Tables 14–29. Key in Appendix F.8.

	View recall	Full search	Partial search	Purchase recall	Item returned
<i>Retail</i>					
Marketplace	-	-	-	-	-
Grocery 1	31.0	89.9	79.9	-	-
Grocery 2	27.1	86.5	59.7	-	-
Pharmacy (browsing)	42.5	80.0	52.2	-	-
Pharmacy (transactions)	-	-	-	20.1	-
Convenience	26.9	-	-	-	-
Home decor	35.6	47.4	29.6	-	-
Camera & photo	-	-	-	35.8	-
<i>Engagement</i>					
Beverage discovery	16.2	89.6	63.1	35.2	95.9
Image discovery	-	-	-	-	-
Short video	1.72	-	-	-	-
Reviews	1.22	-	-	-	-
<i>Payments</i>					
	Merchant		Category		
Cross-merchant	-		-		

Table 15: **BehaviorGPT 12.5B fine-tuned**: recall@10 as a percentage on every cell of the grid, hit@5 in the payments block. **Bold** where it leads every other method in Tables 14–29. Key in Appendix F.8.

	View recall	Full search	Partial search	Purchase recall	Item returned
<i>Retail</i>					
Marketplace	24.0	33.6	24.2	17.4	72.4
Grocery 1	17.4	82.9	69.3	-	-
Grocery 2	13.3	77.1	48.3	-	-
Pharmacy (browsing)	34.7	71.7	44.4	-	-
Pharmacy (transactions)	-	-	-	15.0	-
Convenience	11.0	-	-	-	-
Home decor	26.5	35.7	24.5	-	-
Camera & photo	-	-	-	9.14	-
<i>Engagement</i>					
Beverage discovery	0.602	8.72	2.32	1.52	30.6
Image discovery	0.320	0.310	-	-	-
Short video	0.135	-	-	-	-
Reviews	0.620	-	-	-	-
<i>Payments</i>					
	Merchant		Category		
Cross-merchant	57.7		45.7		

Table 16: **BehaviorGPT 0.5B zero-shot**: recall@10 as a percentage on every cell of the grid, hit@5 in the payments block. **Bold** where it leads every other method in Tables 14–29. Key in Appendix F.8.

	View recall	Full search	Partial search	Purchase recall	Item returned
<i>Retail</i>					
Marketplace	-	-	-	-	-
Grocery 1	20.9	86.3	74.0	-	-
Grocery 2	20.0	82.1	54.1	-	-
Pharmacy (browsing)	38.6	75.4	49.6	-	-
Pharmacy (transactions)	-	-	-	16.3	-
Convenience	22.2	-	-	-	-
Home decor	31.0	42.4	30.0	-	-
Camera & photo	-	-	-	34.8	-
<i>Engagement</i>					
Beverage discovery	6.69	44.5	22.7	8.95	59.2
Image discovery	0.728	0.734	-	-	-
Short video	1.24	-	-	-	-
Reviews	1.01	-	-	-	-
<i>Payments</i>					
	Merchant		Category		
Cross-merchant	-		-		

Table 17: **BehaviorGPT 0.5B fine-tuned**: recall@10 as a percentage on every cell of the grid, hit@5 in the payments block. **Bold** where it leads every other method in Tables 14–29. Key in Appendix F.8.

	View recall	Full search	Partial search	Purchase recall	Item returned
<i>Retail</i>					
Marketplace	0.607	0.425	0.526	2.04	3.62
Grocery 1	13.1	9.13	7.43	-	-
Grocery 2	13.3	7.84	5.99	-	-
Pharmacy (browsing)	29.4	14.6	12.8	-	-
Pharmacy (transactions)	-	-	-	9.10	-
Convenience	8.96	-	-	-	-
Home decor	23.1	19.1	15.0	-	-
Camera & photo	-	-	-	22.0	-
<i>Engagement</i>					
Beverage discovery	0.997	2.03	1.03	2.04	3.06
Image discovery	0.045	0.043	-	-	-
Short video	0.130	-	-	-	-
Reviews	0.303	-	-	-	-
<i>Payments</i>					
	Merchant		Category		
Cross-merchant	9.80		6.90		

Table 18: **SASRec**: recall@10 as a percentage on every cell of the grid, hit@5 in the payments block. **Bold** where it leads every other method in Tables 14–29. Key in Appendix F.8.

	View recall	Full search	Partial search	Purchase recall	Item returned
<i>Retail</i>					
Marketplace	0.563	0.579	0.727	2.03	2.67
Grocery 1	11.4	7.66	6.29	-	-
Grocery 2	11.4	6.48	4.80	-	-
Pharmacy (browsing)	28.4	13.7	12.0	-	-
Pharmacy (transactions)	-	-	-	11.6	-
Convenience	12.5	-	-	-	-
Home decor	21.6	17.2	14.3	-	-
Camera & photo	-	-	-	15.8	-
<i>Engagement</i>					
Beverage discovery	0.918	1.69	1.01	3.36	3.06
Image discovery	0.035	0.033	-	-	-
Short video	0.179	-	-	-	-
Reviews	0.384	-	-	-	-
<i>Payments</i>					
	Merchant		Category		
Cross-merchant	32.9		6.50		

Table 19: **HSTU**: recall@10 as a percentage on every cell of the grid, hit@5 in the payments block. **Bold** where it leads every other method in Tables 14–29. Key in Appendix F.8.

	View recall	Full search	Partial search	Purchase recall	Item returned
<i>Retail</i>					
Marketplace	19.2	9.57	4.06	3.35	14.8
Grocery 1	4.62	2.36	1.77	-	-
Grocery 2	5.61	11.1	0.718	-	-
Pharmacy (browsing)	32.3	20.3	12.6	-	-
Pharmacy (transactions)	-	-	-	6.01	-
Convenience	9.76	-	-	-	-
Home decor	20.0	22.1	15.5	-	-
Camera & photo	-	-	-	10.9	-
<i>Engagement</i>					
Beverage discovery	5.60	5.70	2.77	1.66	13.1
Image discovery	0.088	0.082	-	-	-
Short video	0.437	-	-	-	-
Reviews	0.380	-	-	-	-
<i>Payments</i>					
	Merchant		Category		
Cross-merchant	51.4		33.1		

Table 20: **RecFormer**: recall@10 as a percentage on every cell of the grid, hit@5 in the payments block. **Bold** where it leads every other method in Tables 14–29. Key in Appendix F.8.

	View recall	Full search	Partial search	Purchase recall	Item returned
<i>Retail</i>					
Marketplace	3.78	2.78	1.61	0.768	4.08
Grocery 1	3.08	3.97	2.57	-	-
Grocery 2	1.17	1.96	0.984	-	-
Pharmacy (browsing)	20.7	12.7	12.2	-	-
Pharmacy (transactions)	-	-	-	1.85	-
Convenience	6.97	-	-	-	-
Home decor	10.8	11.8	9.76	-	-
Camera & photo	-	-	-	3.69	-
<i>Engagement</i>					
Beverage discovery	0.258	0.485	0.199	0.548	0
Image discovery	0.135	0.126	-	-	-
Short video	0.087	-	-	-	-
Reviews	0.044	-	-	-	-
<i>Payments</i>					
	Merchant		Category		
Cross-merchant	43.3		23.4		

Table 21: **OneRec 8B Pro**: recall@10 as a percentage on every cell of the grid, hit@5 in the payments block. **Bold** where it leads every other method in Tables 14–29. Key in Appendix F.8.

	View recall	Full search	Partial search	Purchase recall	Item returned
<i>Retail</i>					
Marketplace	8.46	4.43	2.87	0.450	1.57
Grocery 1	1.01	0.748	0.461	-	-
Grocery 2	1.11	0.490	0.244	-	-
Pharmacy (browsing)	21.3	10.2	8.22	-	-
Pharmacy (transactions)	-	-	-	4.58	-
Convenience	5.07	-	-	-	-
Home decor	12.4	13.8	11.1	-	-
Camera & photo	-	-	-	3.64	-
<i>Engagement</i>					
Beverage discovery	0.377	0.355	0.217	0.136	7.14
Image discovery	0.109	0.054	-	-	-
Short video	0.248	-	-	-	-
Reviews	0.402	-	-	-	-
<i>Payments</i>					
	Merchant		Category		
Cross-merchant	48.5		24.1		

Table 22: **RecFormer zero-shot**: recall@10 as a percentage on every cell of the grid, hit@5 in the payments block. **Bold** where it leads every other method in Tables 14–29. Key in Appendix F.8.

	View recall	Full search	Partial search	Purchase recall	Item returned
<i>Retail</i>					
Marketplace	0	0	0	1.24	2.26
Grocery 1	4.27	3.23	2.07	-	-
Grocery 2	3.23	3.23	2.25	-	-
Pharmacy (browsing)	2.69	4.48	3.43	-	-
Pharmacy (transactions)	-	-	-	8.17	-
Convenience	4.04	-	-	-	-
Home decor	4.90	3.19	2.52	-	-
Camera & photo	-	-	-	5.49	-
<i>Engagement</i>					
Beverage discovery	0.292	0.875	0.500	0.146	3.85
Image discovery	0.022	0.027	-	-	-
Short video	0.018	-	-	-	-
Reviews	0.191	-	-	-	-
<i>Payments</i>					
	Merchant		Category		
Cross-merchant	44.7		20.9		

Table 23: **popularity**: recall@10 as a percentage on every cell of the grid, hit@5 in the payments block. **Bold** where it leads every other method in Tables 14–29. Key in Appendix F.8.

	View recall	Full search	Partial search	Purchase recall	Item returned
<i>Retail</i>					
Marketplace	20.3	7.27	4.77	2.64	81.3
Grocery 1	3.82	2.04	1.58	.	.
Grocery 2	0.932	0.514	0.290	.	.
Pharmacy (browsing)	22.3	10.8	9.57	.	.
Pharmacy (transactions)	.	.	.	0.00065	.
Convenience	9.92	.	.	.	.
Home decor	20.6	19.7	14.8	.	.
Camera & photo	.	.	.	3.87	.
<i>Engagement</i>					
Beverage discovery	8.36	2.75	1.01	25.0	42.3
Image discovery	0.518	0.606	.	.	.
Short video	0	.	.	.	.
Reviews	0	.	.	.	.
<i>Payments</i>					
	Merchant		Category		
Cross-merchant	62.3		46.9		

Table 24: **recent items**: recall@10 as a percentage on every cell of the grid, hit@5 in the payments block. **Bold** where it leads every other method in Tables 14–29. Key in Appendix F.8.

	View recall	Full search	Partial search	Purchase recall	Item returned
<i>Retail</i>					
Marketplace	19.2	7.26	4.76	2.76	67.9
Grocery 1	4.13	2.08	1.62	.	.
Grocery 2	1.12	0.560	0.302	.	.
Pharmacy (browsing)	22.1	10.8	9.56	.	.
Pharmacy (transactions)	.	.	.	0.00069	.
Convenience	10.9	.	.	.	.
Home decor	20.2	19.7	14.8	.	.
Camera & photo	.	.	.	4.07	.
<i>Engagement</i>					
Beverage discovery	8.12	2.77	1.02	23.6	42.3
Image discovery	0.520	0.605	.	.	.
Short video	0	.	.	.	.
Reviews	0	.	.	.	.
<i>Payments</i>					
	Merchant		Category		
Cross-merchant	61.0		43.2		

Table 25: **user-popular**: recall@10 as a percentage on every cell of the grid, hit@5 in the payments block. **Bold** where it leads every other method in Tables 14–29. Key in Appendix F.8.

	View recall	Full search	Partial search	Purchase recall	Item returned
<i>Retail</i>					
Marketplace	0	0	0	0	0
Grocery 1	0.174	0	0	-	-
Grocery 2	0	0	0	-	-
Pharmacy (browsing)	0	0	0	-	-
Pharmacy (transactions)	-	-	-	0.00026	-
Convenience	0	-	-	-	-
Home decor	0.265	0	0	-	-
Camera & photo	-	-	-	0	-
<i>Engagement</i>					
Beverage discovery	0	0	0	0	0
Image discovery	0	0	-	-	-
Short video	-	-	-	-	-
Reviews	-	-	-	-	-
<i>Payments</i>					
	Merchant		Category		
Cross-merchant	-		-		

Table 26: **predict-last**: recall@10 as a percentage on every cell of the grid, hit@5 in the payments block. **Bold** where it leads every other method in Tables 14–29. Key in Appendix F.8.

	View recall	Full search	Partial search	Purchase recall	Item returned
<i>Retail</i>					
Marketplace	1.55	1.93	2.41	1.56	2.84
Grocery 1	4.06	41.0	27.6	-	-
Grocery 2	4.05	39.6	22.4	-	-
Pharmacy (browsing)	16.0	42.5	26.5	-	-
Pharmacy (transactions)	-	-	-	5.75	-
Convenience	6.47	-	-	-	-
Home decor	14.1	17.6	10.6	-	-
Camera & photo	-	-	-	10.9	-
<i>Engagement</i>					
Beverage discovery	1.52	13.3	5.93	3.68	0
Image discovery	0.058	0.108	-	-	-
Short video	0.081	-	-	-	-
Reviews	0.465	-	-	-	-
<i>Payments</i>					
	Merchant		Category		
Cross-merchant	39.6		42.3		

Table 27: **Gemma 4 31B**: recall@10 as a percentage on every cell of the grid, ranking a shortlist of the most frequent training products rather than the whole catalog: 1,000 of them over the first 2,000 evaluation sessions on ten rows, and on short video and reviews 1,000 or 167 over every position we score. **Bold** means the same here as in Tables 14–29: this method leads the cell over all sixteen, ours included, and a bold here carries a ~ because the lead rests on a shortlist and on a subsample of our positions. Counts are not inked, every cell but those of the last two rows being such a subsample by construction, which Figure 22 states once instead. That figure’s denominator for a cell is the highest of these three entries, whether or not it is bold. Greyed entries in the search columns are query-blind and barred from the bold. A dash is a cell the arm was never run on. The payments block is the one part of these three tables not on the shortlist protocol: its two tasks predict a merchant or a category, which the tabular pair ranks directly and in full, so those cells are hit@5 over the whole stream and are the numbers Figure 5’s payments block is measured against. Gemma was never run on them.

G Internal Experiments and Ablations

These experiments and ablations, accumulated over the course of development at varying scales and on different internal datasets, motivated the recipe choices of Section 2; Section 3.5 summarizes the ones that bear directly on the paper’s claims.

G.1 Data Mixing Ablation: Setup and Caveats

The base run of Section 3.4 trains on 400M retail and grocery events for five epochs, or 2B events seen. The three 8B arms spend four times that budget: on more retail and grocery data; on the base data plus 1.2B image-discovery events under one shared softmax; and on the same mixture with a separate softmax per distribution.

	View recall	Full search	Partial search	Purchase recall	Item returned
<i>Retail</i>					
Marketplace	0.045	0.094	0.112	0.077	0.184
Grocery 1	0.546	0.498	0.522	-	-
Grocery 2	0.569	0.565	0.497	-	-
Pharmacy (browsing)	0.675	0.530	0.605	-	-
Pharmacy (transactions)	-	-	-	0.494	-
Convenience	0.468	-	-	-	-
Home decor	0.947	1.03	1.32	-	-
Camera & photo	-	-	-	1.97	-
<i>Engagement</i>					
Beverage discovery	0.103	0.208	0.128	0	0
Image discovery	0.0099	0.0092	-	-	-
Short video	0.018	-	-	-	-
Reviews	0.302	-	-	-	-
<i>Payments</i>					
	Merchant		Category		
Cross-merchant	37.6		5.10		

Table 28: **LightGBM**: recall@10 as a percentage on every cell of the grid, ranking a shortlist of the most frequent training products rather than the whole catalog: 1,000 of them over the first 2,000 evaluation sessions on ten rows, and on short video and reviews 1,000 or 167 over every position we score. **Bold** means the same here as in Tables 14–29: this method leads the cell over all sixteen, ours included, and a bold here carries a ~ because the lead rests on a shortlist and on a subsample of our positions. Counts are not inked, every cell but those of the last two rows being such a subsample by construction, which Figure 22 states once instead. That figure’s denominator for a cell is the highest of these three entries, whether or not it is bold. Greyed entries in the search columns are query-blind and barred from the bold. A dash is a cell the arm was never run on. The payments block is the one part of these three tables not on the shortlist protocol: its two tasks predict a merchant or a category, which the tabular pair ranks directly and in full, so those cells are hit@5 over the whole stream and are the numbers Figure 5’s payments block is measured against. Gemma was never run on them.

	View recall	Full search	Partial search	Purchase recall	Item returned
<i>Retail</i>					
Marketplace	0.421	0.161	0.395	0.265	2.39
Grocery 1	0.574	0.547	0.512	-	-
Grocery 2	0.876	0.820	0.502	-	-
Pharmacy (browsing)	1.88	1.06	1.21	-	-
Pharmacy (transactions)	-	-	-	1.05	-
Convenience	1.49	-	-	-	-
Home decor	2.37	1.03	0.732	-	-
Camera & photo	-	-	-	8.81	-
<i>Engagement</i>					
Beverage discovery	0.291	0.848	0.416	0.235	0
Image discovery	0.013	0.022	-	-	-
Short video	0.018	-	-	-	-
Reviews	0.258	-	-	-	-
<i>Payments</i>					
	Merchant		Category		
Cross-merchant	49.7		27.9		

Table 29: **TabPFN v3**: recall@10 as a percentage on every cell of the grid, ranking a shortlist of the most frequent training products rather than the whole catalog: 1,000 of them over the first 2,000 evaluation sessions on ten rows, and on short video and reviews 1,000 or 167 over every position we score. **Bold** means the same here as in Tables 14–29: this method leads the cell over all sixteen, ours included, and a bold here carries a ~ because the lead rests on a shortlist and on a subsample of our positions. Counts are not inked, every cell but those of the last two rows being such a subsample by construction, which Figure 22 states once instead. That figure’s denominator for a cell is the highest of these three entries, whether or not it is bold. Greyed entries in the search columns are query-blind and barred from the bold. A dash is a cell the arm was never run on. The payments block is the one part of these three tables not on the shortlist protocol: its two tasks predict a merchant or a category, which the tabular pair ranks directly and in full, so those cells are hit@5 over the whole stream and are the numbers Figure 5’s payments block is measured against. Gemma was never run on them.

Three caveats bound the result. First, the runs are about two orders of magnitude below full scale, and in the multilingual literature scale relieves interference at fixed capacity [22], so the in-domain cost is plausibly an upper bound. Second, the separate-softmax arm is not a pure control: it also doubles the batch and repeats the retail data for ten epochs rather than five. Third, one engagement source is not a sweep over mixtures; which combinations transfer positively, and how the effect scales, remain open.

G.2 Domains

We find that domains improve the accuracy of the model by about 10% (0.35 to 0.386 in CartMetric accuracy, Appendix D.6), with most gains coming from user behavior with fewer than 50 actions or events.

Feature Removed	Performance Drop
User Type	0.000511
Hour of Day	0.000811
Device Type	0.002612
City	0.003444
Weekday	0.011484
Day of Month	0.049557
Month	0.241633
Year	0.350336
<i>All Features (Baseline)</i>	0.000000

Table 30: Ablation study showing the performance drop when removing individual features from a full model. The full model uses all features; performance is measured relative to this baseline.

Domains can differ significantly across datasets and settings; some domains exist only in certain datasets. We therefore remove domains that are too narrow in the first stage of training, so the model cannot identify the setting of the data from the domains and overfit to it. See Table 31 for results.

Training Setup	Validation Loss
Keep all domains in Stage 1 and Stage 2	5.635
Remove narrow domains in Stage 1 and Stage 2	5.574
Remove narrow domains in Stage 1, add all domains in Stage 2	5.546

Table 31: Effect of removing narrow domains on model performance. Validation loss is reported after Stage 2 training. Narrow domains are defined as those highly specific to individual datasets and are removed to prevent overfitting.

Overfitting to domains. Domains are a real overfitting risk, especially time domains. At one point we were getting biased results that occurred only for a specific month and year—which, on inspection, was also when the items in question had been interacted with; there must have been a campaign in just that month and year. Such mappings from domains to actions are easy for the model to find and overfit to, and considerable effort goes into preventing it.

G.3 Embedders

Source embedder helps. We first studied the impact of different embedders on a dataset of around 600M e-commerce events. This is a relatively small dataset, and the generalization was in-domain. We trained with stage 1 (embedder trainable) and stage 2 (embedder fixed). See Table 32 for results.

Target embedders. Using the same feature source embedder, we evaluate different target strategies; see Table 33. Here we train on two similar domains rather than a single one, so the ability to use multiple domains matters more, and other target objectives begin to gain an edge. We train stage 1 until validation loss increases. Next, after some optimizations, we also implemented MLM-Description, randomly masking events and generating their text descriptions from the surrounding events. However, MLM converged after 66 epochs versus 16 for CLM—more than $4\times$ the compute—and did not scale well to larger datasets. See Table 34.

Model Variant	NLL ↓	Notes
Source One-Hot	6.849	Baseline using one-hot source encoding
Source Embedder	6.231	Learns source embeddings; adds params/compute

Table 32: Comparison of negative log-likelihood (NLL) between source encoding methods. The source embedder substantially improves performance, but it also has higher parameter and compute cost. Data set was fixed.

Stage 2 Objective	NLL ↓	Notes
CLM Objective	7.069	Baseline using one-hot target encoding
Keyword Prediction (CE)	7.003	Predicts event keywords via cross-entropy
CLM-Description	6.924	Generates next event descriptions; more params/compute

Table 33: Stage 2 performance (negative log-likelihood) on data under different stage 1 training objectives. CLM-Description performs best despite increased computational cost. Data set was fixed. CLM-Description is more than $10\times$ slower to train than the baseline and Keyword-Prediction.

When event frequencies are more long-tailed—most actions appearing very rarely while a few dominate—source and target feature embedders help even more; see Table 35. Here CLM-Description (with a source embedder) outcompetes the one-hot approach at both equal data and equal compute time: just because the actions themselves are long-tail does not mean their features are.

Pre-trained Embedders. A frozen pre-trained image embedder reaches a loss of 6.515, versus 6.491 for an image embedder trained end-to-end. End-to-end training is more sensitive to the quality and correctness of the images, however: removing low-quality images and training end-to-end on the rest led to a loss of 6.364. See Table 36 for results.

On text the effect is much larger. On the search and recommendation task, a text encoder trained end-to-end on behavior reaches a loss of 7.0 after one epoch, against 8.03 for an encoder initialized from a BERT pretrained on generic Swedish and 8.00 for one pretrained on product descriptions and prices (Table 37). The margin over pretraining is close to 1.0 in loss, more than an order of magnitude larger than the 0.024 separating frozen from end-to-end on images. The two pretrained initializations are meanwhile indistinguishable from each other, which we attribute to a corpus mismatch: the product-pretraining data consisted of descriptions and prices unlike the descriptions seen during fine-tuning, so in-domain pretraining bought nothing over generic language pretraining.

Image transformations can translate to text. In a joint embedding space such as CLIP, a transformation learned on one modality may carry to the other. We learn a matrix M from user behavior at a merchant so that $X_{img}MX_{img}^T$ is the better image metric, and apply it as $X_{text}MX_{img}^T$ to tune text-to-image retrieval (search) to the same merchant. It transfers: Table 38 shows large gains for search as well.

Cache embeddings (MoCo). Caching candidates from the two previous batches, in the style of MoCo [44], slightly improved our stage-1 model (Table 39).

Softmax both directions. CLIP applies its contrastive softmax in both directions. We tried the same; it did slightly better in-domain but worse out-of-domain. See Table 40.

Size of embedding vectors. One way to ease the bottleneck of embedding large vocabularies is to make the embeddings smaller. Table 41 shows one example on e-commerce data: reducing the embedding size from 768 to 64 hurts performance overall. Evaluated on low-frequency events alone, the differences are even larger (Table 42), so the smaller size hurts rare events most.

Quantizing image embeddings. In Table 43, we show an evaluation of our VQVAE on image embeddings. We start from image embeddings that live in a joint text-and-image latent space. For each dataset we look at image and text embeddings: the baseline retrieves, for each event’s text, the events with the closest images; vqvae-text does the same but with quantized image embeddings; and vqvae-img retrieves, for each (non-quantized) image embedding, the closest ones in the quantized space. Scores are top-10 hit rate, across several data sources. We later verified that performance in the training pipeline is not hurt on pre-training or downstream tasks.

Objective	NLL ↓	Epochs to Converge	Notes
CLM-Description	6.36	16	Efficient convergence
MLM-Description	6.33	66	Slower; 4× more compute; poorly scales

Table 34: Comparison between CLM- and MLM-based description generation. While MLM achieves slightly better NLL, it requires significantly more training epochs and does not scale well to larger datasets.

Model Variant	NLL (Same Data) ↓	NLL (Same Time) ↓
CLM One-Hot	6.262	6.197
CLM-Description (with Embedders)	6.194	6.188

Table 35: Comparison under long-tail event frequency: CLM-Description with learned source/target embedders consistently outperforms one-hot encodings under both equal data size and equal compute time.

Prices. We show results on including prices, either as exact floating-point numbers or bucketed into integer intervals. See Table 44 for results across two merchants. On this dataset we train for 5 epochs, so there is a risk of overfitting—remembering products by a specific price rather than generalizing—and the results reflect it.

Popularity & trends of interactions. Incorporating the popularity or trending aspect of events and actions is critical: we want to understand whether, say, a product or post is trending and how that affects a user’s propensity to act on it—i.e., which users are more or less trend-sensitive. Table 45 reports performance across two markets when including the change in number of interactions with events as targets. It comes at a $2\times$ training cost.

G.4 Candidate Spaces

Batch candidates and sampled candidates. When we are training over a large set of possible candidates and cannot afford to embed all possible next actions, we need to sample. First, we already have embedded the set of all batch events, so we might as well use them as candidates. In addition, we might want to uniformly randomly sample from the full vocabulary, to get a larger space of candidates (if compute allows). In Table 46, we show results of adding random samples to the batched candidates, with and without freezing the target embedder. The best results come from adding random samples with a trainable target embedder. Since training with random samples brings the setup closer to stage 2 (which trains only on random samples), we also compare regular stage 2 against training batch + random samples to convergence—and batch + random samples won out, though barely.

Scaling number of candidates. As we scale to larger candidate spaces (vocabularies), it becomes more and more important to sample a larger number of candidates during training. Stage 1 and stage 2 thus reinforce each other, and stage 2 becomes about sampling as many candidates as possible.

Table 47 shows the difference in performance, now on a larger behavioral dataset with a much larger vocabulary (close to 1B), between sharding 8M candidates so that the softmax is computed separately over each 1M candidates per device versus jointly across all 8M on all devices. It also shows how important stage 2 is to training—freezing the embedder and maximizing candidates converges later—and that using LoRAs to push the number of stage-2 candidates even further also helps.

Separate distributions. Mixing candidate distributions is unproblematic when the whole batch draws from the same distribution; problems arise when different distributions are combined within a single batch, where the model can cheat by learning to distinguish the distributions rather than the targets. Fully disjoint target distributions can also be suboptimal: we want the model to learn to compare things transitively. If it learns to compare popular products, and learns to compare some of those popular products with unpopular ones, it has thereby learned to compare both. For example, with one distribution of popular products and one per category, a model deployed over all products can reason: from popular products, the next target should be a popular product within the shoes category; from comparisons within the shoes category, the target should most likely be this specific product. Through the shared items, what the model learns on each intersection lets it compare across the union. Table 48 shows that 24K random candidates together with batched ones (but with separated losses) performs better than 24K or even 48K random candidates without separation.

Image Embedder Variant	NLL ↓	Notes
Pre-trained (Frozen)	6.515	No fine-tuning; fixed visual features
End-to-End (All Images)	6.491	Trained from scratch; sensitive to image quality
End-to-End (Filtered)	6.364	Trained after removing low-quality images

Table 36: Performance comparison of different image embedder training strategies. End-to-end training outperforms pre-trained embeddings, but is highly sensitive to image quality. Filtering low-quality data yields the best result.

Text Encoder	Loss ↓	Notes
End-to-end on behavior	7.0	No language pretraining
Pretrained, generic Swedish	8.03	One epoch on the search/rec task
Pretrained, product descriptions and prices	8.00	Pretraining corpus differs from fine-tuning text

Table 37: Text encoder training for the search and recommendation task, after one epoch. Training end-to-end on behavior clearly outperforms either pretrained initialization. The two pretrained variants are indistinguishable, the product-pretraining corpus having differed from the descriptions used for fine-tuning.

Qualitatively, we also find that separate softmaxes generalize trends and patterns much better: the model does not separate the embedding spaces but learns to align them, which helps it use data from other sources and merchants to generalize to new merchants and domains.

Label Smoothing. While we saw some positive effects of label smoothing in smaller-scale experiments, the benefits did not persist as we scaled.

G.5 Fine-tuning

LoRA helps generalization. We noticed that increasing the candidate space in stage-2 can lead to overfitting to that particular vocabulary. We therefore compared training only LoRA adapters in stage 2 with training the full network. See Table 49 for results.

Pretraining versus training from scratch. Figure 26 gives one case of what pretraining is worth before any fine-tuning budget is spent (Section 3.2).

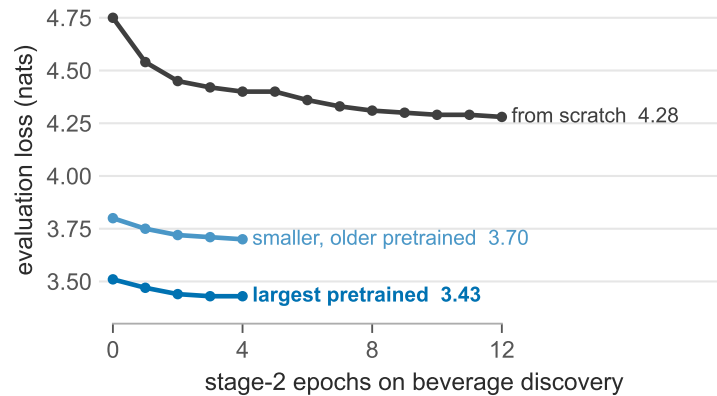


Figure 26: **Pretraining is worth more than a longer fine-tune.** Evaluation loss on beverage discovery during stage 2, by starting point, in one example run. From scratch, twelve epochs end 0.85 nats above the largest pretrained checkpoint after four.

Model	Loss ↓	Search Score ↑	Cart Score ↑
Baseline (Identity Transformation)	10.235	0.062	0.145
Learned Transformation	6.603	0.246	0.220

Table 38: Comparison of baseline vs. learned transformation on search and cart metrics. The learned model significantly improves both scores and reduces test loss.

Caching Strategy	NLL (Stage 1) ↓
Without Cache	6.248
With Cache (MoCo-style)	6.231

Table 39: Effect of using MoCo-style cache embeddings in Stage 1. Caching candidates from two previous batches yields a small improvement in negative log-likelihood.

G.6 Decoding

Greedy vs Beam. We find that beam search is less helpful in decoding actions than it is for language, compared to greedy decoding. Table 50 gives one example. We believe this is because actions are less strictly ordered than language, which has a more formal grammar.

G.7 Augmentations

Randomly dropping events. We tried randomly dropping events, but it was hard to get it to contribute. Table 51 shows one such comparison at a 10% drop rate, which helps slightly for merchant 2.

Where to make use of LLMs. Using an LLM to generate augmentations of item metadata, such as alternative descriptions, improved generalization.

G.8 Reasoning

With the procedure of Appendix B.8, test-time compute improved clear tasks such as predicting the next purchase after a session of viewing products, from 13% accuracy with SFT to 25% with GRPO.

The universal reasoning method requires more compute, but with modest resources it increased accuracy universally across tasks, from 8.9% (SFT) to 9.7% (GRPO).

See Table 52 for a summary of results.

G.9 Scaling Up

Pretraining across datasets. The earliest evidence that pretraining on several datasets helps comes from an older model: pretraining stage 1 on two grocery datasets and running stage 2 on only one of them improved that dataset’s loss, and visibly its search and recommendations, over pretraining on it alone. Including the second dataset in stage 2 as well helped only slightly, 6.552 against 6.557 nats of test loss once both are scored with the same output layer and context length. Removing the domains only one dataset carries—year, city, user type and device—improved stage 2 further (Table 31), since otherwise the model can recognize which dataset a sequence came from instead of learning the behavior the datasets share.

For scaling up with multiple datasets from different sources, Table 53 gives an example of how stage 1 performs, how stage 2 adds performance, and how separating the distributions adds further still. Lastly, we show improvements from carefully locating correspondences between actions from different data sources, to identify which are actually the same even when described differently.

H A/B Test Details

What the commercial comparisons support. The commercial baselines in Table 4 were licensed, indexed, configured, and tuned by the partner rather than by us, so a row is evidence about one production deployment of a system at one

Contrastive Objective	In-Domain Loss ↓	Out-of-Domain Loss ↓
Normal Direction Only	6.990	9.208
Softmax Both Directions	6.921	9.594

Table 40: Effect of using bidirectional softmax (as in CLIP) during contrastive training. While in-domain performance improves, out-of-domain generalization worsens.

Model	Merchant 1				Merchant 2			
	Overall	Partial	Search	Product	Overall	Partial	Search	Product
Size 768	6.062	5.847	2.724	8.150	6.398	6.675	2.915	7.196
Size 64	6.984	7.363	2.704	8.144	6.723	8.008	2.856	7.324

Table 41: Evaluation on Merchant 1 and Merchant 2 test sets. Exp-1 uses the full dataset; Exp-4 uses a mini dataset with Stage 1 pretraining. Metrics include overall, partial search, full search, and product-level losses.

point in time, including that partner’s catalog, ranking rules, and manual tagging, and not about the vendor’s product in general. We anonymize them because naming the vendor would invite the stronger reading and would in most cases identify the partner. Each incumbent is also the version live when the test ran, between 2020 and 2026, not necessarily the product sold under that name today.

Engagement: featured content in a niche-interest app. This 26-day test is the only one in Table 4 with day-level results rather than a single period average. Featured content ranked by BehaviorGPT drew 11.7% more clicks than the control, and those clicks led to 19.5% more orders: the click-to-order conversion rate rose from 1.93% to 2.07%, a 7% relative improvement, and $1.117 \times 1.07 \approx 1.195$. The variant led on clicks on all 26 days and on post-click orders on 22 of them. Under the null of no difference, each day is a fair coin, giving one-sided sign-test p -values of $2^{-26} \approx 1.5 \times 10^{-8}$ for clicks and $p \approx 3 \times 10^{-4}$ for orders. Days share users and are therefore not strictly independent, so we read these as indicative rather than exact.

Model	Overall ↓	Partial Search ↓	Search ↓	Product ↓
Size 768	6.453	6.287	3.514	7.925
Size 64	7.387	7.746	3.625	8.001

Table 42: Performance on low-frequency (≤ 100) events in the Merchant 1 test set. Size 768 outperforms Size 64 even more than when evaluated on all frequencies.

Table 43: Evaluation of VQVAE image embedders on various datasets. Scores are top-10 accuracy.

Model	groceries			art			pharmacy			general retail		
	baseline	vqvae-img	vqvae-text	baseline	vqvae-img	vqvae-text	baseline	vqvae-img	vqvae-text	baseline	vqvae-img	vqvae-text
VQVAE-300M	0.4828	0.9900	0.0863	0.5103	0.9466	0.2739	0.8031	0.9963	0.2479	0.5296	0.9982	0.4209
VQVAE-4B	0.4828	0.9966	0.1648	0.5103	0.9999	0.4114	0.8031	1.0000	0.4560	0.5296	0.9995	0.4554
VQVAE-4B-small	0.4828	0.9973	0.2233	0.5103	0.9998	0.4318	0.8031	1.0000	0.5771	0.5296	0.9995	0.4672

Model Variant	Merchant 1 Test				Merchant 2 Test			
	Overall	Partial	Search	Product	Overall	Partial	Search	Product
baseline	6.062	5.847	2.724	8.150	6.398	6.675	2.915	7.196
floats	6.042	5.795	2.727	8.194	6.378	6.686	2.880	7.171
buckets	6.098	5.872	2.764	8.209	6.372	6.636	2.896	7.172

Table 44: Comparison of models trained with price or cost features on Merchant 1 and Merchant 2 test sets. Lower is better. Neither variant shows a clear advantage.

Model Variant	Market 1 Loss	Market 2 Loss
Baseline	2.69	4.49
+ Interactions as Targets	2.57	4.40

Table 45: Impact of incorporating interaction trends (i.e., change in number of interactions) on prediction performance across two markets. Lower values indicate better loss. Including interaction dynamics improves results.

Model Variant	Overall ↓	Partial Search ↓	Search ↓	Product ↓
batch + train target	5.804	4.942	2.416	7.001
batch + 3K random + train target	5.402	4.539	2.192	6.555
batch + 8K random + train target	5.327	4.463	2.136	6.475
batch + 16K random + train target	5.280	4.420	2.103	6.425
batch + frozen target	6.219	5.355	2.634	7.464
batch + 8K random + frozen target	5.660	4.783	2.274	6.862
batch + 16K random + train target (conv.)	5.072	4.285	1.962	6.169
batch + 16K random + train target + stage-2	5.076	4.282	1.967	6.175

Table 46: Performance on the e-commerce test set using different sampling strategies and target embedder configurations. Incorporating random candidates improves performance, with the best results from training with 16K random candidates until convergence.

Model Variant	Epoch	Loss ↓
Stage 1	20	6.219
Stage 2 (8M, 1M per softmax)	1	5.074
	2	5.085
Stage 2 (8M, all in one softmax)	1	4.825
	2	4.844
	3	4.815

Table 47: Loss values across Stage 2 training setups using retail dataset. Sharded candidate strategies with a unified softmax achieve the best loss at epoch 3.

Extra Candidates	Overall ↓	Partial Search ↓	Search ↓	Product ↓
0 (baseline)	7.234	7.123	4.850	8.439
24,000 random	6.724	6.624	4.461	7.856
48,000 random	6.494	6.431	4.250	7.519
24,000 random + separation	6.381	6.344	4.190	7.314

Table 48: Performance impact of increasing the number of extra candidates during training. Adding more candidates consistently improves results, with the best performance achieved when separating the losses for the two distributions (random and batched)

Model Variant	Merchant 1				Merchant 2 (OOD)			
	Overall	Partial	Search	Product	Overall	Partial	Search	Product
Baseline Full	4.682	4.171	2.258	7.022	5.767	5.430	2.693	6.641
LoRA	4.729	4.242	2.312	7.008	5.658	5.176	2.558	6.582

Table 49: Stage 2 with a fully trainable model (Baseline) versus LoRA. LoRA does better on the OOD dataset, which is from a different merchant than the one the model was trained on.

Decoding Method	Accuracy ↑	Notes
Beam Search (beam=7)	0.088	Sequence length = 10
Greedy Decoding	0.097	Higher accuracy despite no search

Table 50: Accuracy of predicted user actions using different decoding strategies. Greedy decoding outperforms beam search in this setup.

Model	Merchant 1				Merchant 2			
	Overall	Partial	Search	Product	Overall	Partial	Search	Product
Baseline	6.062	5.847	2.724	8.150	6.398	6.675	2.915	7.196
Drop 10%	6.069	5.848	2.737	8.167	6.372	6.667	2.880	7.167

Table 51: Comparison of Stage 1 models with and without event sampling. Event sampling shows slight trade-offs: slightly worse in Merchant 1 overall/search but marginally better in Merchant 2 overall/product. This is test loss, lower is better.

Table 52: Accuracy improvements from using test-time computation and GRPO for behavioral reasoning tasks.

Setting	Method	Accuracy (%)	Improvement
Next-purchase after viewing session	SFT	13.0	–
	GRPO	25.0	+12.0
Universal reasoning across tasks	SFT	8.9	–
	GRPO	9.7	+0.8

Model Variant	Overall	FullSrch	PartSrch	Brand	Cat	ViewProd	<sep>	<purchase>	<refund>	Merchant	PurchProd	RefProd
Stage 1	6.027	4.351	6.181	4.620	5.228	5.815	7.767	7.738	0.729	8.473	10.419	5.666
+ Stage 2	5.501	3.897	5.583	4.521	4.949	5.696	7.049	7.104	0.636	8.020	10.019	4.787
+ Separation	5.386	3.792	5.451	4.438	4.917	5.612	6.936	7.204	0.677	7.950	10.042	4.722
+ Correspondences	4.995	3.370	4.994	4.162	4.480	5.486	6.696	4.915	0.190	7.242	9.574	3.500

Table 53: Comparison of model variants trained with different augmentation strategies on event-level metrics. All values represent loss; lower is better across all columns.

I Qualitative Examples from the Demos

This appendix shows screenshots of three demos built on the SDK (Appendix J.1): the e-commerce storefront (Appendix I.1), Curate my wall, an art recommender (Appendix I.2), and a label-free fraud detector (Appendix I.3).

I.1 Storefront

The figures in this subsection are screenshots of the storefront demo (Section 4), an application built on the SDK, serving the BehaviorGPT 0.5B and 12.5B checkpoints over a general marketplace retail assortment of several million items. Each screenshot has the same layout. On the left is the storefront, whose search results and recommendations are the model’s ranking of the full catalog given the session so far. On the right is the session drawn as a sequence of events, with the model’s most likely next items listed at the bottom. Red annotations mark what changed between panels. Where we have the same scenario on the 12.5B checkpoint (Figures 28, 30 and 33), we show it as well: the behaviors carry over to the larger model. The demo labels that checkpoint *V4.0-13B*, or *V4.0-12.5B* in the later screenshots.

Nothing in these examples is a rule written for the demo. Each change in the ranking comes from the model reading something different in its input: a context attribute (date, gender, market), an earlier event in the session, or the query text itself.

Context attributes. Date, gender and market enter the model as contextual attributes of the session (Section 2.2), and changing any one of them, with the query held fixed, changes the ranking. Figure 27 changes only the session date: the query *chocolate* returns everyday chocolate on an ordinary day, heart-shaped boxes on Valentine’s Day and holiday gift boxes on Christmas Eve. The model adapts to every date of the year in this way; these two holidays simply make the effect easy to see. The 12.5B shows the same shift (Figure 28): heart-shaped boxes and chocolate-dipped strawberries on Valentine’s Day, holiday gift boxes on Christmas Eve, and neither on an ordinary day. Figure 29 does the same for gender (*pants*) and market (*spade*); in the Swedish market the ranking shifts to Swedish-listed garden tools. The 12.5B reads the market the same way (Figure 30): *spade* returns Kate Spade handbags in the United States and garden spades in Sweden.

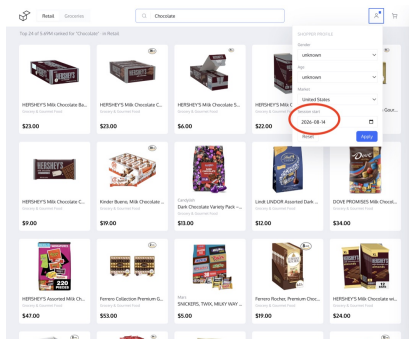
Behavioral context. Figure 31 shows how earlier events in the session change what a query means. Searching *case* with no history returns the most popular iPhone cases; the same search after a search for *Samsung* returns Samsung cases. In a second session, a Samsung phone is added to the cart, and the recommendations then anticipate what goes with it: a matching case after the phone, earbuds after the charger. The basket in panel (h)—phone, case, screen protector, charger and earbuds—was assembled by following these recommendations.

Figure 32 combines an earlier session with the date. A shopper searches for baby-shower items in February, adds two of them and checks out. With the session date moved to late October, the home page leads with Halloween costumes for babies, followed by strollers, cribs and baby monitors: from a baby shower eight months earlier and the approaching holiday, the model infers that the household probably now has an infant. The 12.5B, given the same session in April (Figure 33), draws the same inference: on 30 October its home page is given over to infant products, a bassinet, diapers, formula and toddler toys, with Halloween costumes, one of them for a baby, among the clothing.

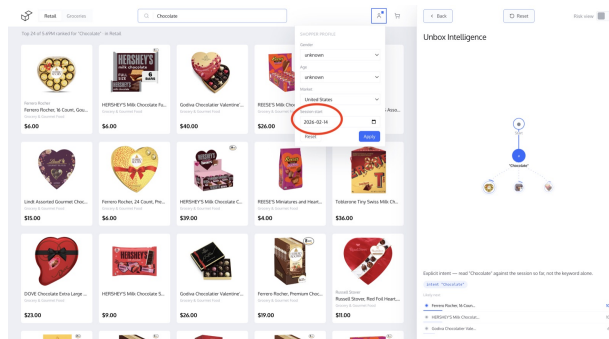
Reading the query. Figure 34 shows two properties of search. The query enters the model as text and is read against the session, not matched against a keyword index, so a misspelling such as *plystton* still returns PlayStation products, on both the 0.5B and the 12.5B. The same query also changes meaning with the session: *brush* with no history returns mostly hair brushes and styling tools, while after a foundation is added it returns cosmetics brushes first, here with the ranking grouped into themes.

I.2 Curate my wall

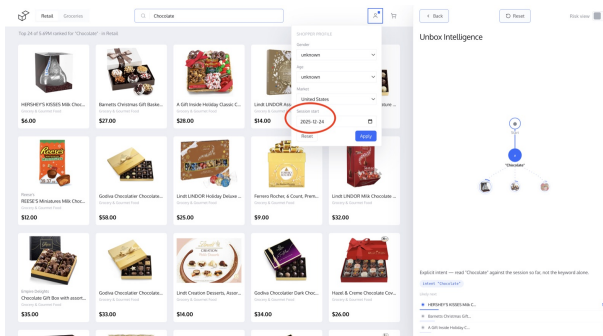
We embedded 19,000 public-domain artworks from the Art Institute of Chicago as a BehaviorGPT catalog, served by the 12.5B checkpoint with no interaction data and no task-specific training; it can be tried through <https://www.unboxai.com/behaviorgpt>. Each action a visitor takes—viewing a work, hanging it on a personal wall, taking it down, or searching—is appended to their history as an event, and recommendations and search results are re-ranked from that history in real time. Adding 19,000 H&M products to the same catalog lets the same art-only history recommend fashion: kimonos and printed scarves for a visitor drawn to Japanese prints, beaded jewelry for one drawn to ancient Egypt, and color-blocked knits and abstract prints for one drawn to Kandinsky (Figure 35). No event in that history involves clothing. We include this example as an illustration of what can be built on the SDK, not as an evaluation.



(a) An ordinary date.

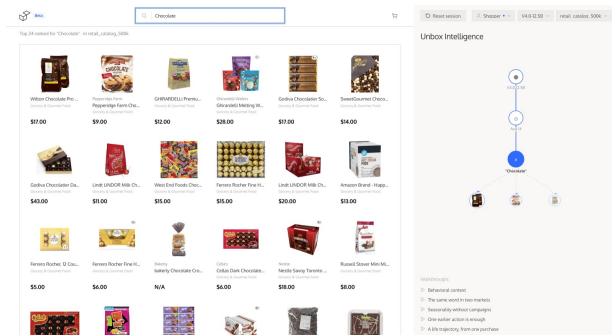


(b) Session date set to 14 February.

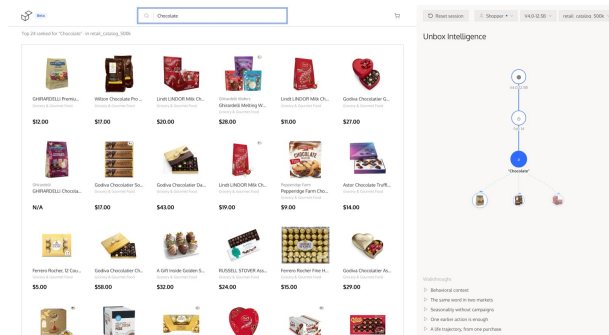


(c) Session date set to 24 December.

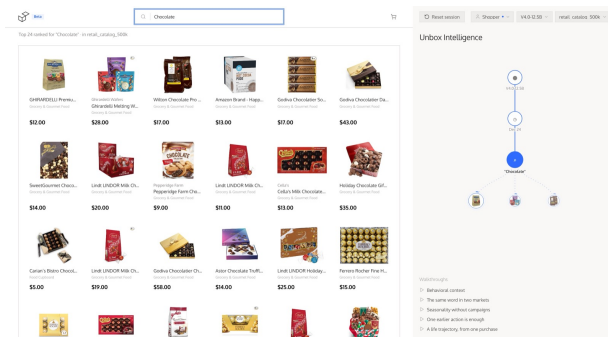
Figure 27: The same query, *chocolate*, on three dates. Only the session date changes between panels.



(a) Session date set to 14 August.

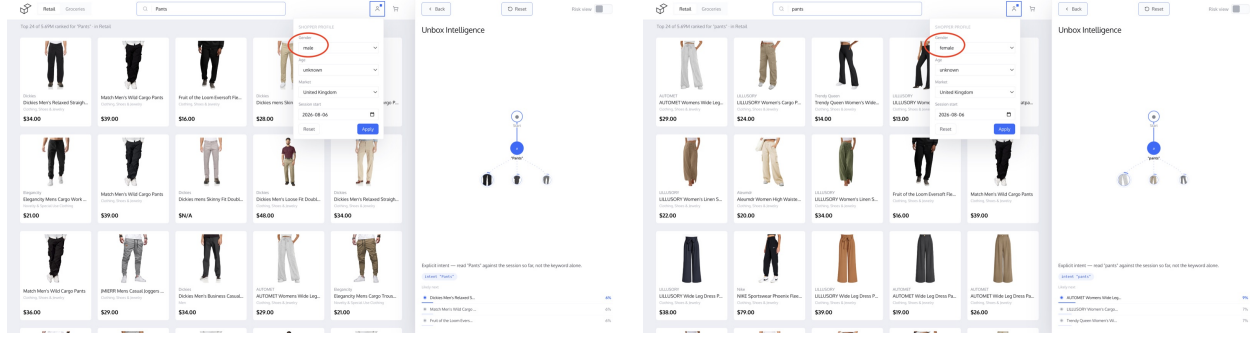


(b) Session date set to 14 February: heart-shaped boxes.



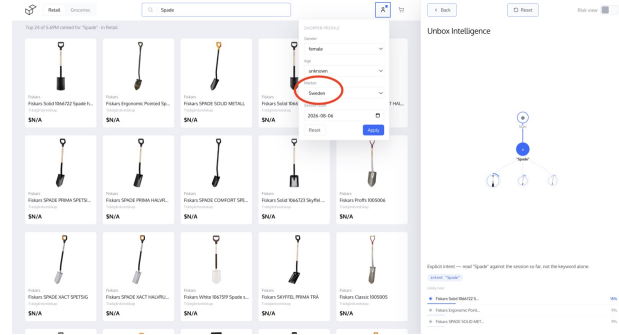
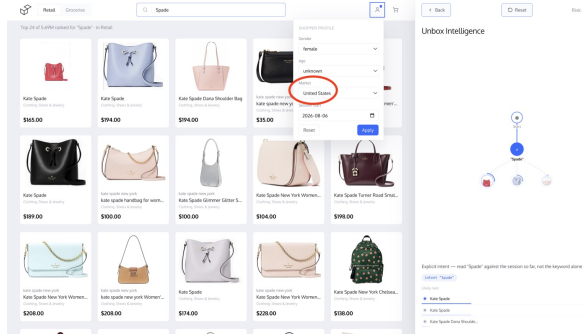
(c) Session date set to 24 December: holiday gift boxes.

Figure 28: The same date effect on the 12.5B checkpoint (labeled V4.0-12.5B in the demo). Only the session date changes between panels.



(a) *Pants*, gender set to male.

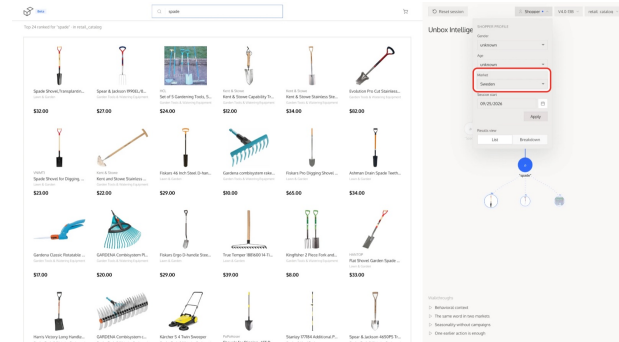
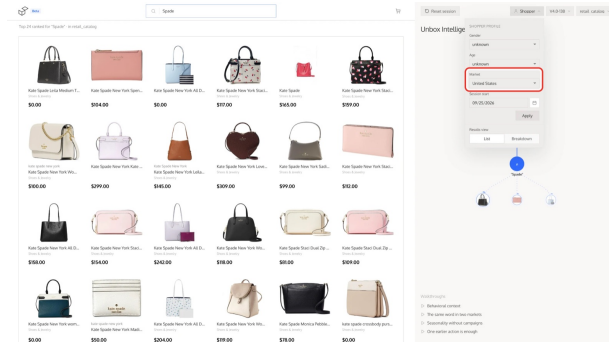
(b) *Pants*, gender set to female.



(c) *Spade*, United States market.

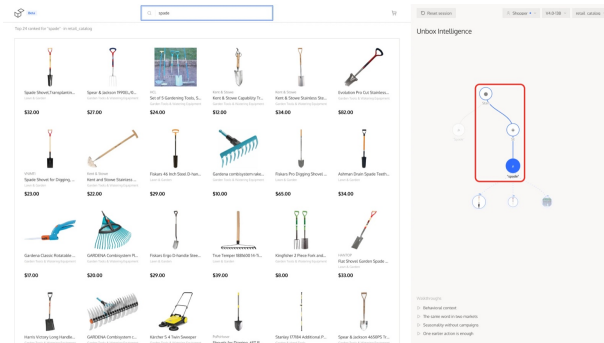
(d) *Spade*, Swedish market.

Figure 29: **Gender and market as context.** Top: the same query under two genders. Bottom: the same query in two markets.



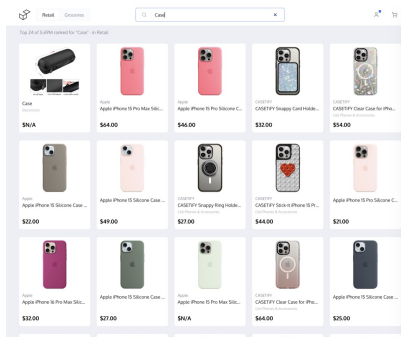
(a) *Spade*, United States market: Kate Spade handbags.

(b) *Spade*, Swedish market: garden spades.

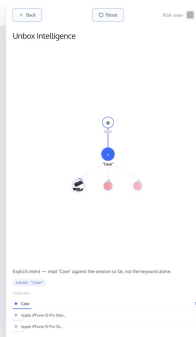


(c) The Swedish session drawn as a sequence of events.

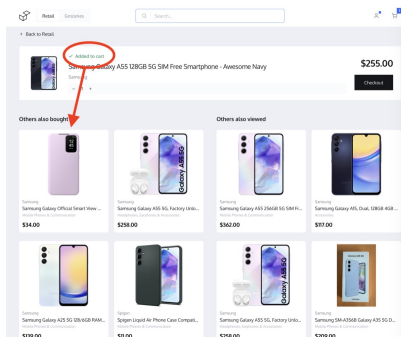
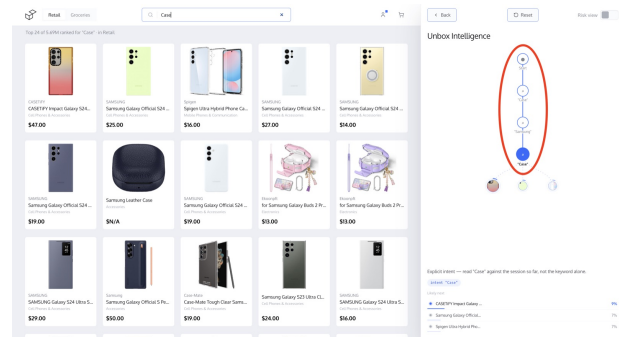
Figure 30: **The same market effect on the 12.5B checkpoint** (labeled V4.0-13B in the demo). Only the market changes between (a) and (b).



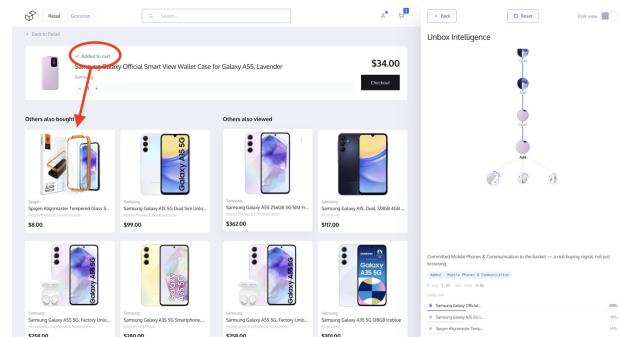
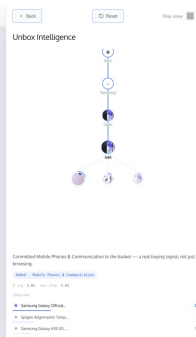
(a) Case with no history: iPhone cases.



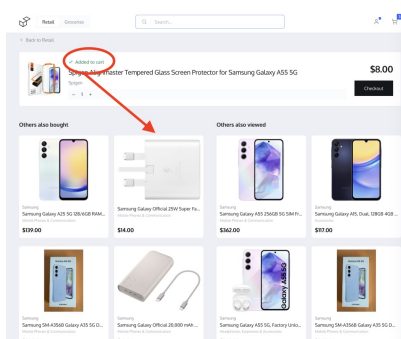
(b) Case after searching Samsung: Samsung cases.



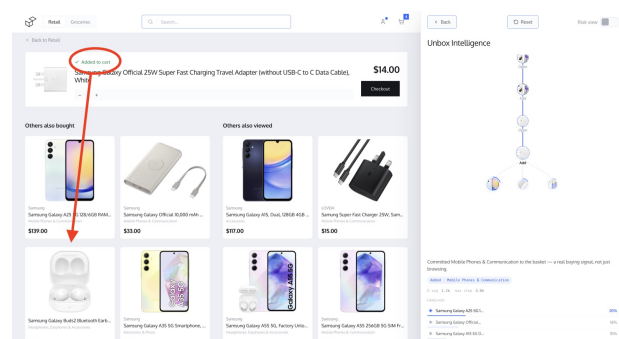
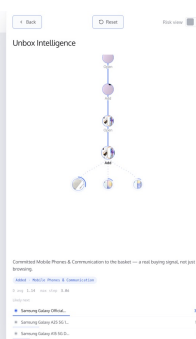
(c) New session: a Samsung phone added; a matching case is recommended.



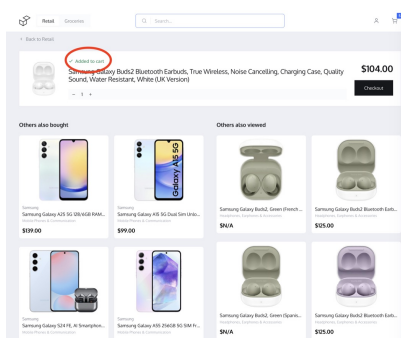
(d) The case added.



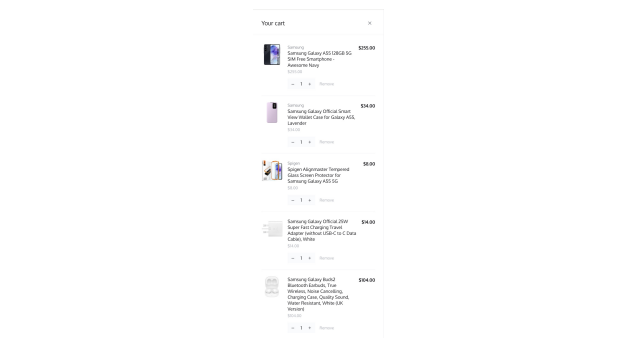
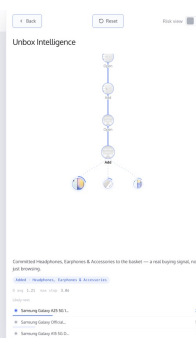
(e) A screen protector added.



(f) A charger added; earbuds are recommended.

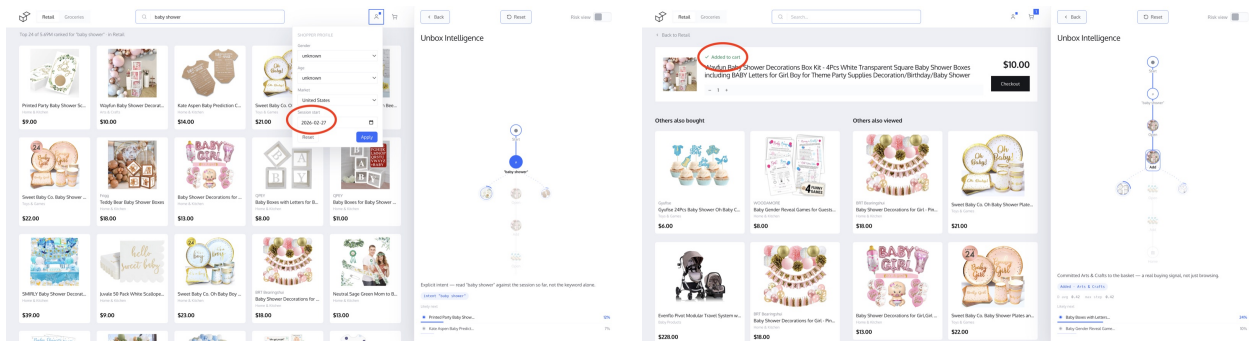


(g) The earbuds added.



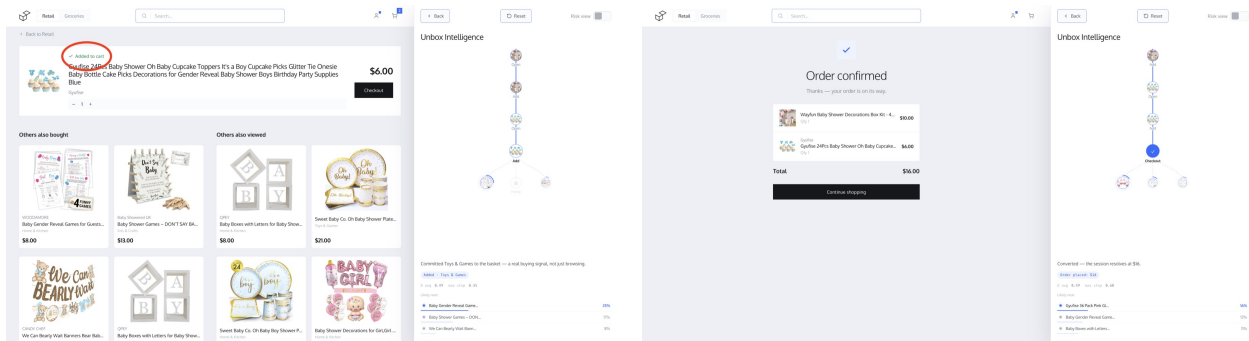
(h) The resulting basket.

Figure 31: Earlier events change what a query means, and recommendations anticipate the next purchase. (a–b) The same query with and without an earlier search. (c–h) One session built by following the recommendations after a phone is added.



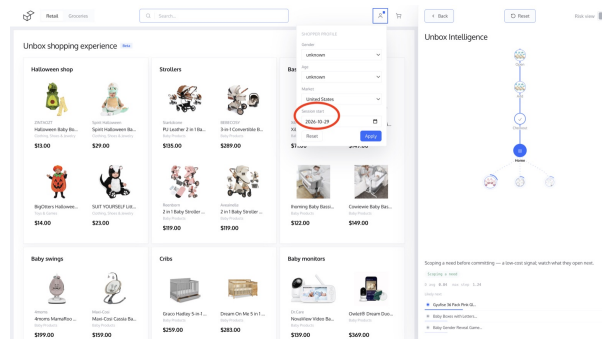
(a) Searching *baby shower* in February.

(b) A decoration added.



(c) A second baby-shower item added.

(d) Checkout.



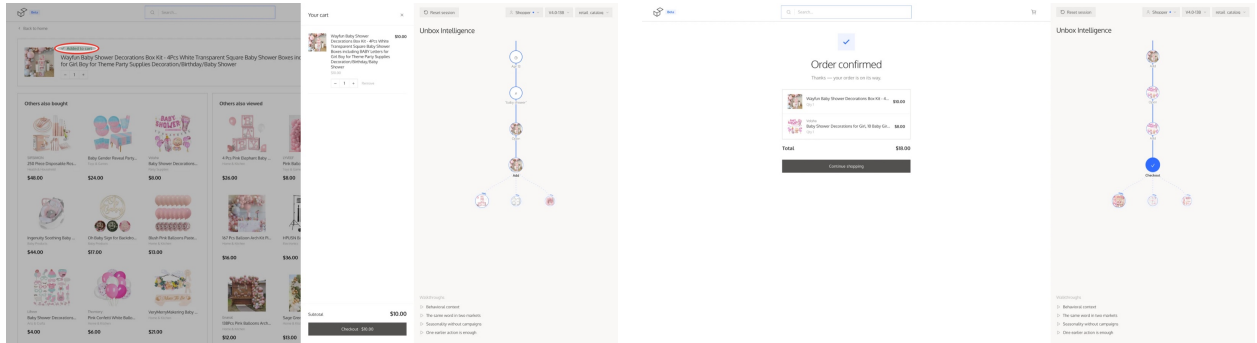
(e) Session date moved to 29 October: baby Halloween costumes lead the home page.

Figure 32: **An earlier session and the date combined.** A baby-shower purchase in February and a date in late October together produce recommendations for an infant at Halloween.

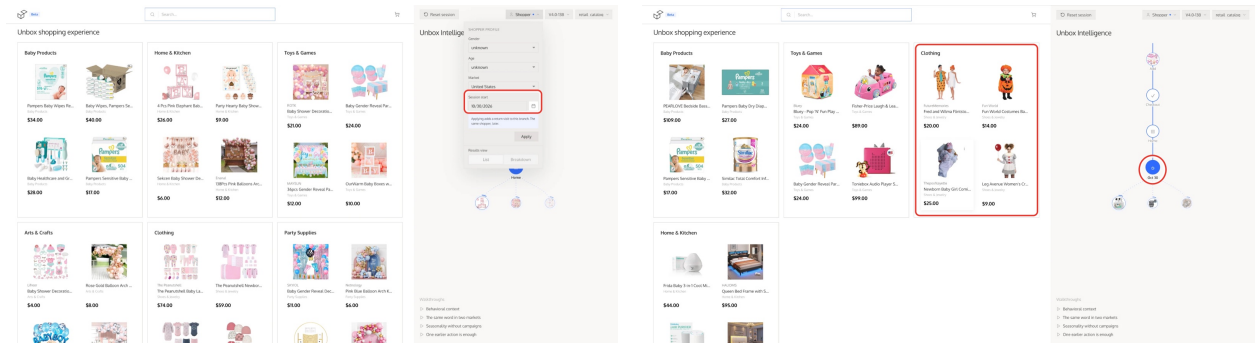
I.3 Behavioral anomaly detection

BehaviorGPT also detects anomalous behavior, with no task-specific training or labels. We built a fraud-detection demo on the 12.5B checkpoint on a public retail catalog. Each action in a session (a search, product view, add-to-cart or order) is scored by its surprisal under the model given the session so far, in excess of the model’s own uncertainty at that step: an action the model expects scores low, one it cannot explain scores high, and an action is not penalized merely for following an ambiguous history. The scores accumulate over the session (a CUSUM statistic), and the session is flagged once the sum crosses a threshold set to flag 2% of normal shoppers. Because each score belongs to one action, every flag comes with its explanation: the actions that raised it and the items the model expected instead.

Figure 36 shows an account takeover. The owner searches *headphones*, views four pairs and adds one to the cart; from step 7, someone else adds a Pixel 7 Pro phone, a gas lawn mower and a crossbow, then checks out. The owner’s six steps stay within 1.5σ of normal shoppers, the three intruding ones score 5.3 – 6.0σ , and the session crosses the threshold at step 8, before the order is placed. The phone accounts for 32% of the excess surprisal: it ranks 13,740th of 94,162 items for this history, where the model expected headphones. The lawn mower accounts for 31%, where the model, now reading the phone as part of the session, expected another Pixel phone. On simulated sessions at a 2% false-alarm rate, the detector caught account takeovers, card testing, scraper bots and click fraud, usually within one to three steps of the first fraudulent action. The next-action prediction that serves recommendations thus also serves as a likelihood model of behavior that needs no labels. As with Curate my wall, we include the demo as an illustration, not as an evaluation (Section 5).

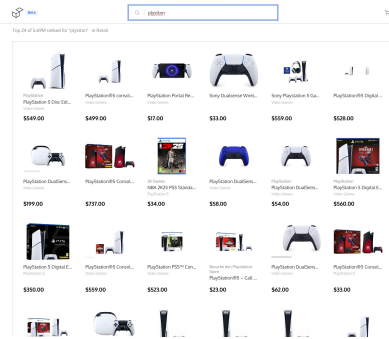


(a) Session date 13 April: *baby shower* searched and a decoration added. (b) A second item added, then checkout.

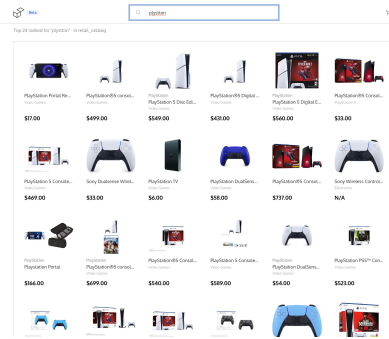
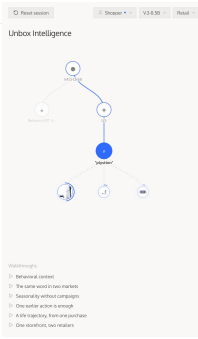


(c) Session date moved to 30 October. (d) The home page on 30 October: infant products, and Halloween costumes among the clothing.

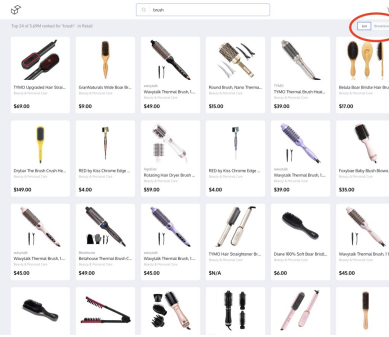
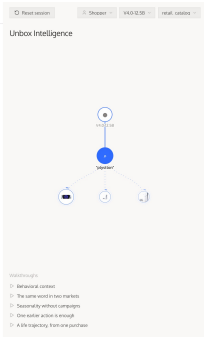
Figure 33: **The same session on the 12.5B checkpoint** (labeled *V4.0-13B* in the demo). A baby-shower purchase in April and a date in late October produce recommendations for an infant, with Halloween costumes alongside.



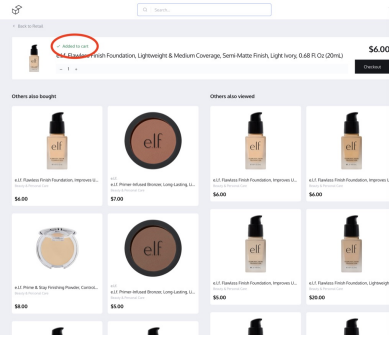
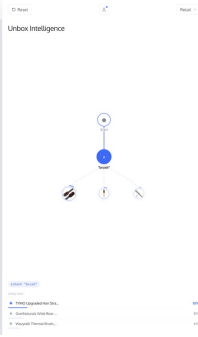
(a) *Plyston* on the 0.5B.



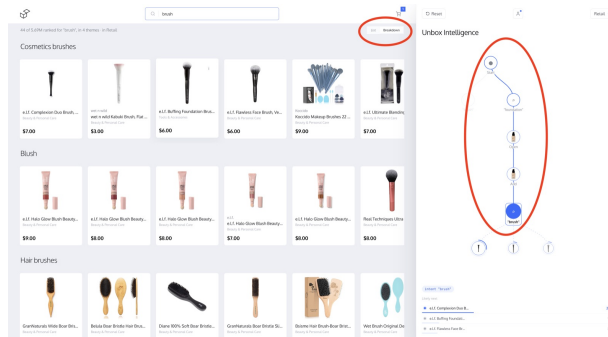
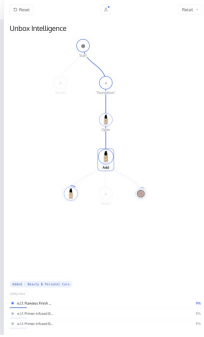
(b) *Plyston* on the 12.5B.



(c) *Brush* with no history.

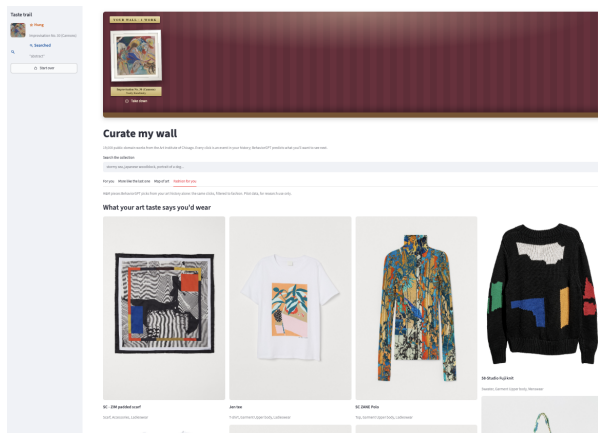


(d) A foundation searched for and added.

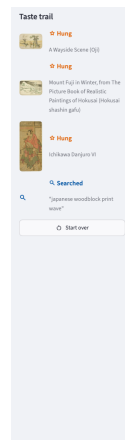


(e) *Brush* again, grouped into themes: cosmetics brushes first.

Figure 34: **Reading the query.** (a–b) A misspelling is resolved from the query text by both checkpoints. (c–e) The same query before and after a related purchase, the second grouped into themes.

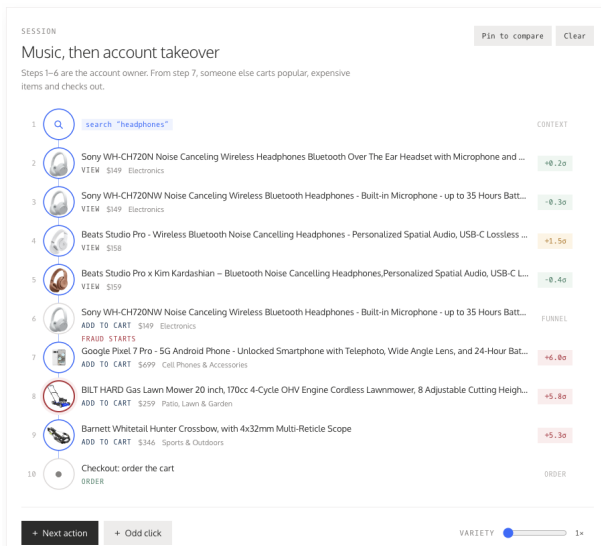


(a) A search for “abstract” and Kandinsky’s *Improvisation No. 30* (*Cannons*) hung: a color-blocked scarf, a printed tee, an abstract-print top and a color-blocked knit.

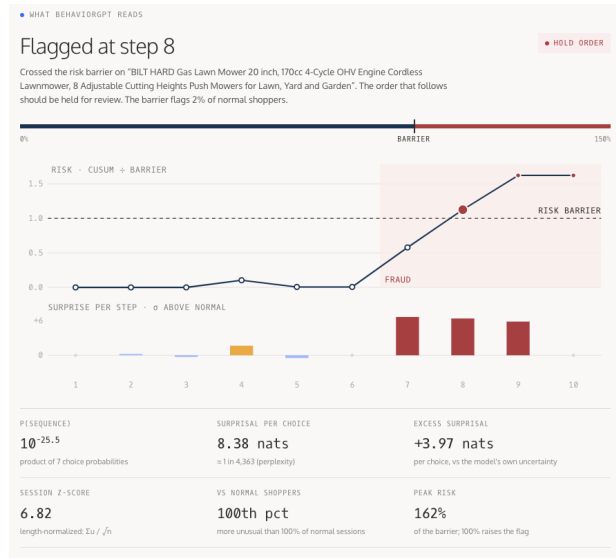


(b) A search for “japanese woodblock print wave” and three Hokusai prints hung: printed scarves, a tiger-print tee and a kimono.

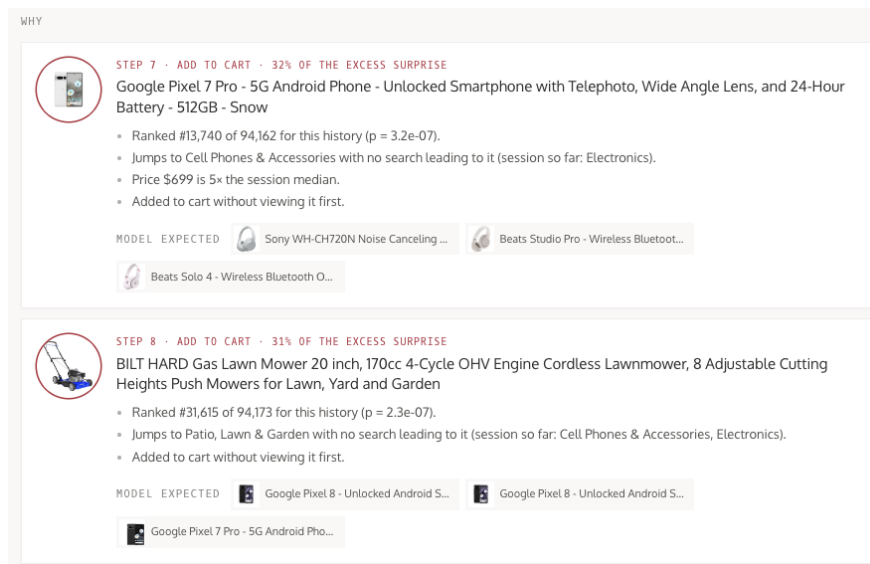
Figure 35: **Fashion recommended from a history of art alone.** The Curate my wall demo on the 12.5B checkpoint. In each panel the visitor’s history (left) holds only art events, and the *Fashion for you* tab ranks the H&M items in the same catalog from that history.



(a) The session, with each step's surprise in standard deviations above normal shoppers. Steps 1–6 are the account owner; steps 7–10 are not.



(b) Accumulated risk relative to the threshold, which flags 2% of normal shoppers. The session crosses it at step 8, before checkout.



(c) The explanation: the two steps that contribute most to the excess surprisal, and the items the model expected at each.

Figure 36: **An account takeover flagged without labels.** The fraud-detection demo on the 12.5B checkpoint on a public retail catalog.

J Applications

Each application below points to where it is already shown, in this paper or in our earlier reports.

- **Recommendations.** A/B tested against an internal recommender, a most-bought baseline and a visual-art engine (Table 4; 36, 35), and measured offline as next-item recall in Figures 5 and 11. Similar and complementary items separate because item embeddings are learned from what is bought together rather than from how items are named [36].
- **Search.** A/B tested against four commercial search and discovery vendors, with search-term suggestions raising interactions by 49.1% (Table 4; 36, 35); refined search terms suggested across some fifty languages [35]; the full- and partial-search columns of the breadth matrices; and query understanding in the demo, including misspellings (Appendix I).
- **Keywords, themes, and categories.** Dynamic categorization A/B tested at +11%, and generated keywords that expert labellers missed [35]; personalized themes in the demo (Appendix I); the objective in Appendix B.7; and next-category prediction on the payments stream (Figure 5).
- **Returns.** The item-returned column of the breadth matrices (Figures 5 and 11).
- **Fraud.** A model of $p(\text{behavior sequence})$ flags sequences that are highly improbable, such as at self-checkout [36]; the corresponding task is behavioral risk scoring (Table 5).
- **Cybersecurity & defense.** Detecting synthetic or adversarial behavior, such as malicious bots, simulating real users, and forecasting action trajectories in operational settings [33].
- **Loyalty programs, marketing, and coupons.** Individualized intent prediction in place of top-down segmentation, and end-to-end engagement optimization [33].
- **Assortment optimization.** Store embeddings clustered for regional assortment, +2.2% in-store sales [36], and gaps in an assortment filled for an estimated +14% in sales [35] (Table 4).
- **Business intelligence.** Customer segments a consultancy study missed [36], country-level structure in employee embeddings [34], brand propensities read directly off the model (Appendix J.2), and the analysis uses of Appendix A.4.

User embeddings. Clustering users on the output distribution over their full history works poorly, as the output distribution is dominated by the most recent context. Instead, we consider representations across the network’s layers: different layers offer different perspectives. Concretely, we add a CLS token with a special position that participates even during next-event prediction, providing global information at different stages (i.e., at different layers of the network). Who a user *is* spans a spectrum from stable long-term traits to current impulses and environmental conditions, so collapsing a user to a single simple description can be counterproductive. Projections of such user embeddings have been shown for shoppers [36] and for employees [34].

J.1 Applications built on the SDK

Because the SDK (Section 4) deploys a catalog zero-shot, an application built on it needs no interaction data of its own. The live demos are built this way; we describe two, and invite others to build their own.

E-commerce storefront. The main live demo, shown in Appendix I, is an e-commerce storefront: its search results and recommendations all come from SDK calls against a deployed catalog, with no logic written for the demo.

Curate my wall. An art recommender over 19,000 public-domain artworks from the Art Institute of Chicago, built the same way on the 12.5B checkpoint with no interaction data, shown in Appendix I.2.

J.2 Behavioral analytics by querying the model

A generative model of behavior answers questions a ranker cannot be asked. Demographic and calendar attributes enter as domains (Section 2.2) and every event type shares one output space, so $p(\text{target} \mid \text{context})$ can be read off for any context: a month, an age band, a merchant, a search term, a product, or a whole sequence of actions. We illustrate it with two brands, a broadband provider and a soft drink, asking what precedes a purchase of each. The readings are aggregate conditional probabilities of BehaviorGPT 0.5B; no individual-level record is accessed or reported.

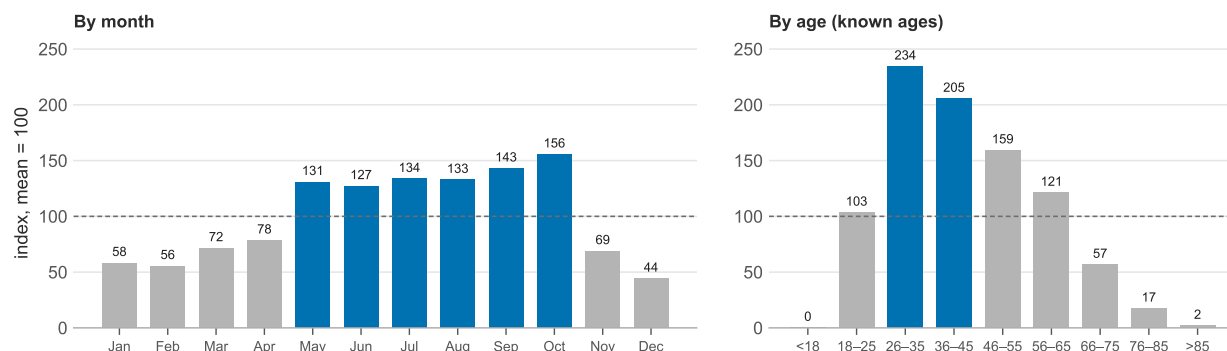


Figure 37: **Propensity for a broadband brand, read off the model by month and by age.** Each bar is $p(\text{brand} \mid \text{domain})$ as an index against the mean over the bars in its panel, 100 being the mean; ages with no recorded value are excluded.

Broadband. Propensity is elevated from May to October, at $2.18\times$ the November–April level, and peaks in October at $3.5\times$ December (Figure 37); by age it peaks at 26–35, at $2.3\times$ the mean over age bands, and falls steadily after 45. The associations say more. Beyond the direct signals—the brand’s own hardware, mobile hotspots and mesh Wi-Fi, competing carriers—the non-network products that rank highest are household staples: laundry detergent, paper goods, cleaners, baby wipes, alongside moving and home-setup merchants such as U-Haul, Home Depot and IKEA. No description of a detergent relates it to broadband; the behavior does, through the move or new household in which both are bought, which is the lactose-free milk of Section 1 on a commercial question. Scoring whole sequences makes the same point at the level of a trajectory: sequences of competing-carrier activity score highest, at a mean p of 1.5×10^{-3} and $7.6\times$ the lowest segment, electronics purchases, while a sequence of nothing but electricity-provider lookups scores 7.8×10^{-4} , above most shopping sequences, the model reading a move from two events.

Beverage. The associated products rank the brand’s own variants first, at a mean p of 0.14, then competing sodas at 0.071, energy drinks, teas and juices, waters, and snacks at 0.013. Among search terms that do not name the brand, *beer* scores 0.093, as a mixer, and *hamburger* 0.041 and *pizza* 0.018, as meal pairings. By month, May carries a lift of 2.07 against October’s 0.54.

What these readings are. They are the model’s conditional probabilities, not causal effects, and they have not been checked against either brand’s own sales. They inherit the coverage of the pretraining data: demographic fields are missing for most users, so only recorded values are reported, and a correlation the sampling produces the model reproduces. We present them as a capability and as hypotheses for a brand owner to test, not as findings about its customers.