Discussion Paper Series

IZA DP No. 18429

February 2026

Earning While Learning: How to Run Batched Bandit Experiments

Jan Kemper
University of Mannheim and ZEW

Davud Rostam-Afschar
University of Mannheim, IZA@LISER, GLO and NeSt

IZA DP No. 18429 February 2026

Earning While Learning: How to Run Batched Bandit Experiments

Abstract

Researchers typically collect experimental data sequentially, allowing early outcome observations and adaptive treatment assignment to reduce exposure to inferior treatments. This article reviews multi-armed-bandit adaptive experimental designs that balance exploration and exploitation. Because adaptively collected experimental data through bandit algorithms violate standard asymptotics, inference is challenging. We implement an estimator that yields valid heteroskedasticity-robust confidence intervals in batched bandit designs and compare coverage in Monte Carlo simulations. We introduce bbandits for Stata, a tool for designing experiments via simulation, running interactive bandit experiments, and implementing and analyzing adaptively collected data. bbandits includes three common assignment algorithms — ε -first, ε -greedy, and Thompson sampling — and supports estimation, inference, and visualization.

JEL classification

C1, C11, C12, C13, C15, C18, C8, C87, C88, C9, D83

Keywords

randomized controlled trial, causal inference, multi-armed bandits, experimental design, machine learning

Corresponding author

Davud Rostam-Afschar
rostam-afschar@uni-mannheim.de

1 Introduction

Randomized controlled trials remain the gold standard for causal inference, but fixed and balanced assignment can be costly when treatments directly affect participants' productivity, health, or welfare. In such settings, one-shot experiments expose too many participants to inferior treatments, incurring financial, ethical, or opportunity costs. If data arrive sequentially—as is common in practice—researchers can instead use early outcomes to adapt treatment assignment, improving participant welfare while still learning about treatment effects. Applications range from pharmaceutical dosing and medical procedures to labor market programs, ad campaigns, pricing policies, and portfolio choice. Adaptive designs help reduce the ex-post regret of allocating participants to inferior treatments when this information could have been learned earlier.

When treatments or participants are costly, traditional designs may also be infeasible because too few observations per arm prevent exploration of options that look less promising ex ante. Even when the goal is purely ex-post inference, fixed budgets and time horizons often require researchers to preselect a small set of treatments, which is problematic in settings with many possible dosages, advertising variants, or vignette profiles. Adaptive designs can deliver conclusive causal evidence quickly and allow treatment choices to be determined by data rather than assumptions.

Most sequential designs can be implemented as multi-armed bandits: a decision maker repeatedly chooses among actions ("treatment arms") with unknown reward distributions and faces a trade-off between exploring all arms and exploiting the best-known option. Bandit algorithms balance this trade-off by minimizing regret. Regret is the loss from not always choosing the best arm. Bandit algorithms trading off exploration and exploitation have been widely studied in statistics, machine learning, and increasingly in applied fields such as medicine, economics, political science, and education. Examples include (Bibaut and Kallus 2025; Hirano and Porter 2025; Chen and Andrews 2023; Caria et al. 2023; Hadad et al. 2021; Avivi et al. 2021; Zhan et al. 2021; Athey and Imbens 2019; Hoffmann et al. 2023; Camerer et al. 2024; Chapman et al. 2024) for economics, (Offer-Westort et al. 2021) for political science, (Gaul et al. 2025) for survey methods research, (Lei et al. 2022) as an example for medicine, (Rafferty et al. 2019) for education, (Schulz et al. 2020) for psychology, etc. The advantages of such algorithms have also led to a rapid growth of interest by practitioners (Hill et al. 2017; Scott 2015; Agarwal et al. 2014; Chapelle and Li 2011; Scott 2010; Graepel et al. 2010).

In this paper, we make three contributions: i) we provide an accessible introduction to treatment assignment using batched adaptive experiments, ii) we introduce bbandits, a new Stata package for designing, running, and analyzing batched bandit experiments, and iii) we offer guidance and best practices for applied researchers based on Monte Carlo results.

More specifically, we analyze this exploration–exploitation trade-off using popular sequential assignment algorithms, including ε-first, ε-greedy, and Thompson sampling. Because adaptive sampling leads to unequal assignment probabilities, classical inference fails. We study experiments conducted in batches and show how valid causal inference

can be obtained using batched OLS (BOLS) in the case of heteroskedasticity (Kemper and Rostam-Afschar 2025; Hirano and Porter 2025; Zhang et al. 2020). A Monte Carlo study evaluates the performance of BOLS relative to the usual OLS estimator across different batch sizes, numbers of batches, and effect sizes.

These simulations reflect scenarios common in applied work. We find that inference in adaptive experiments requires sufficiently large batch sizes—around 20 observations per batch in a two-armed design. The number of batches matters much less. Usual confidence intervals or bootstrap methods may perform poorly when true effects are small and overreject the hypothesis of no difference between arms. This is because the asymptotic variance depends on the treatment assignment probability that does not concentrate. Intuitively, if differences in outcomes across treatments are very small, a natural consequence of experiments with small changes in doses or granular interaction terms, it is hard to decide which arm to assign over the other. BOLS corrects this issue (Kemper and Rostam-Afschar 2025; Zhang et al. 2020). In practice, both estimators yield similar point estimates, even in small samples. We recommend computing both OLS and BOLS and using BOLS for small margins.

Moreover, adaptive experiments routinely generate heteroskedastic per-batch variances—for example, when treatment probabilities become highly imbalanced, when outcomes are binary with different success probabilities, or when one treatment induces inherently higher variance (such as riskier investments or more variable health responses). Under such conditions, the homoskedastic BOLS statistic is invalid and leads to systematic size distortions. In contrast, the heteroskedastic BOLS statistic remains asymptotically normal and delivers correct rejection rates across all designs. For this reason, researchers should always use heteroskedasticity-robust BOLS.

Finally, we introduce the Stata command **bbandits**, which integrates python routines to provide a simple interface for simulating, running, and analyzing batched bandit experiments. It implements ε-first, ε-greedy, and Thompson sampling; supports Monte Carlo design; and offers valid inference, diagnostics, and visualization tools for adaptively collected data.

In the next section, we introduce the ε-greedy and Thompson sampling algorithms and describe the data structure of batched experiments. In Section 3 , we discuss batched regression as a method for correct causal inference. In Section 4, we show results from Monte Carlo simulations to assess the properties of the BOLS. In Section 5, we describe the syntax and options of the new **bbandits** command and provide a step-by-step guide for how to run batched bandit experiments. We provide empirical applications in Section 6 and show how to run own bandit experiments with **bbandits** in section 7. Section 8 concludes.

2 Methods

2.1 Adaptive experimental designs and algorithms

We consider a finite set of treatment arms $k \in \{1, \ldots, K\}$. For each experimental unit i, let $A_i \in \{1, \ldots, K\}$ denote the assigned arm, and let Y_i be the observed outcome (e.g., success vs. failure, or a continuous response). In batched experiments, units are grouped into batches indexed by $t = 1, \ldots, T$, and we write $A_{i,t}$ and $Y_{i,t}$ for unit i in batch t.

In some applications, there is only one observation per batch. In most real world applications, however, batch sizes can be substantial, e.g., more than 40,000 participants in a clinical trial. The batch size is typically set to be constant or random.

Consider a policy maker that asks a researcher to design an experiment to find out which of several information campaigns about vaccines effectively reduces infections. The question is now how to learn which arm reduces infections most effectively without exploring treatment arms that are less effective too often.

In many applications, treatments are costly to administer or units are scarce, so researchers face a trade-off between *exploration* (learning which arms have high expected outcomes) and *exploitation* (assigning more units to arms that already look promising). Multi-armed bandit algorithms provide data-driven treatment assignment rules that explicitly manage this exploration–exploitation trade-off.

We focus on three popular algorithms: ε-first, ε-greedy and Thompson sampling. These algorithms are well established in the reinforcement learning literature. A classical reference is Sutton and Barto (2018). For an overview, see Burtini et al. (2015).

These can be implemented at the unit of observation level or at the batch level, in which observations are grouped. In the batched setting, the assignment in batch $t + 1$ is based only on the information up to batch t for all units.

ε-first (explore-then-commit). The ε-first strategy is closely related to classical A/B testing. Fix a fraction $\varepsilon \in (0, 1)$ of the total number of observations to be used for exploration. In the exploration phase, units are assigned uniformly at random to each arm. Let $\widehat{\mu}_k^{\text{explore}}$ denote the sample mean outcome for arm k over the exploration phase:

$$\widehat{\mu}_k^{\text{explore}} \;=\; \frac{1}{N_k^{\text{explore}}} \sum_{i:A_i=k} Y_i,$$

where N_k^{explore} is the number of observations assigned to arm k during exploration.

In the remaining $(1 - \varepsilon)$ fraction of the experiment, the algorithm *commits* to the arm with the highest empirical mean, assigning all units to

$$k^* \;=\; \arg\max_k \widehat{\mu}_k^{\text{explore}}.$$

Example of explore-then-commit: successive arm-elimination. For cases in which there are two arms and the number of batches is small (2-5) and fixed, Perchet et al. (2016) compare several algorithms to optimally chose the batch sizes. Esfandiari et al. (2021) extend the results of Perchet et al. (2016) to more than two arms and provide a simple rule for successive elimination of arms. In batch t, each active arm is pulled roughly q^t times, where $q = N^{1/T}$ increases geometrically with the batch number. This schedule ensures that early batches are small (allowing coarse but inexpensive estimation), while later batches grow large enough to sharply distinguish the remaining arms.

After each batch, the algorithm computes the empirical mean of each active arm and removes those that are demonstrably worse than the current best arm by more than a confidence buffer. Because each arm is sampled q^t times by the end of batch t, the confidence intervals tighten at a geometric rate, allowing the algorithm to eliminate bad arms early while ensuring the optimal arm is not mistakenly removed.

ε-greedy. The ε-greedy algorithm interleaves exploration and exploitation over the entire experiment. A *greedy* policy is a myopic rule that always selects the arm with the largest current estimate of the expected outcome, ignoring any value of information from further exploration. ε-greedy perturbs this purely greedy rule by adding random exploration.

Let $\varepsilon_t \in [0, 1]$ be the exploration probability in batch t and let $\widehat{\mu}_{k,t}$ be the empirical mean of arm k based on all data up to batch t:

$$\widehat{\mu}_{k,t} \;=\; \frac{1}{N_{k,t}} \sum_{(i,s):\, s \le t,\, A_{i,s}=k} Y_{i,s},$$

where $N_{k,t}$ is the number of times arm k has been played up to and including batch t.

For each unit in batch $t+1$, ε-greedy assigns: with probability $1 - \varepsilon_t$, the *greedy arm*

$$k_t^* = \arg\max_k \widehat{\mu}_{k,t},$$

and with probability ε_t, one of the arms k is chosen uniformly at random.

If ε_t is constant, the algorithm continues to explore forever, which ensures some robustness but prevents convergence to always playing the empirically best arm. A common choice is to let ε_t *decrease over time* (e.g. $\varepsilon_t \propto 1/t$), so that the algorithm explores more in early batches and gradually exploits more as information accumulates.

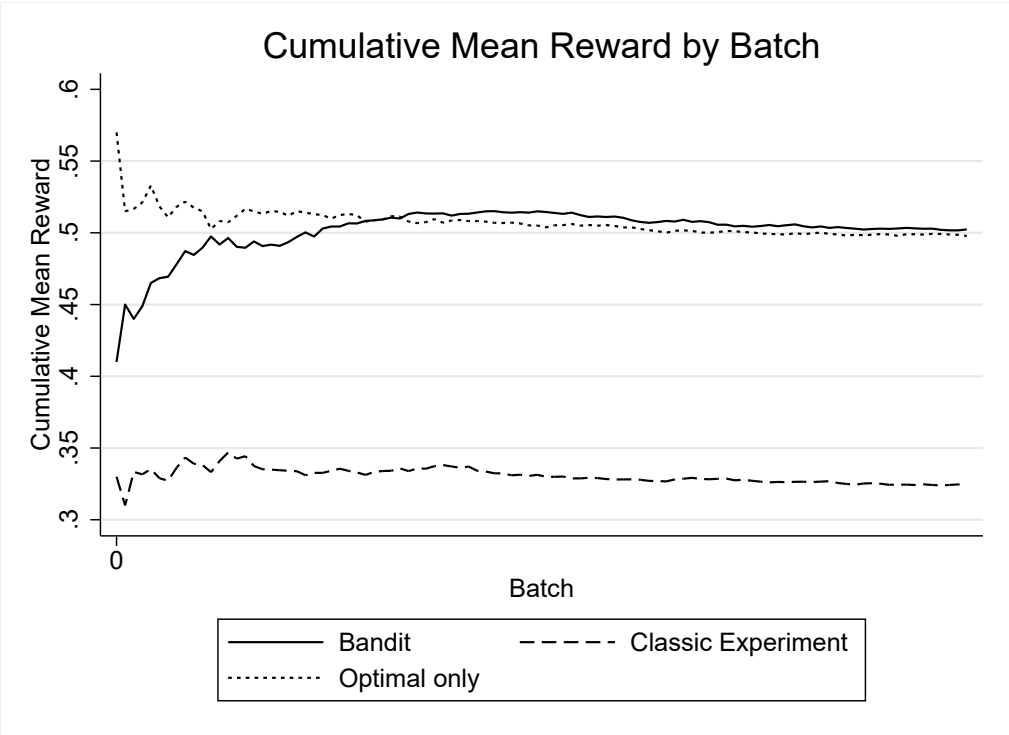
Compared to ε-first, ε-greedy typically achieves a more balanced trade-off between exploration and exploitation when the time horizon is uncertain or when nonstationarity is a concern. When rewards are stationary and the horizon is known ex ante, ε-first may be more efficient because it concentrates exploration in an initial phase.

Thompson sampling. A classical heuristic known as Thompson sampling was developed for allocating experimental effort in bandit problems arising in clinical trials (Thompson

1933, 1935). Thompson sampling models uncertainty about the shape of the distribution from which the observed outcomes are drawn and the expected outcome explicitly.

Figure 1 illustrates the cumulative mean outcome of always playing the optimal arm (oracle benchmark), Thompson sampling, and a traditional fixed balanced design. Thompson sampling sacrifices some reward to explore but substantially less than a non-adaptive design.

Figure 1: Cumulative mean outcome under Thompson sampling vs. fixed design



Notes: The simulated experiment consists of three arms with true success probabilities 0.5, 0.4, and 0.3. The Bernoulli Thompson sampling algorithm with a clipping rate of 0.05 and a decay rate of the clipping rate of 0.9 was applied. The simulation underlying the figure was generated by `bbandits_sim 0.5 0.4 0.3, size(100) batch(100) clipping(0.05) decay(0.9) thompson`.

For Bernoulli outcomes (e.g., success vs. failure), we assume that for each arm k, outcomes are conditionally independent drawn from

$$Y_i \mid A_i = k \sim \text{Bernoulli}(\theta_k),$$

where $\theta_k \in [0, 1]$ is the unknown success probability of arm k.

Let $\mathcal{H}_t = \{(A_{i,s}, Y_{i,s}) : s \leq t\}$ denote the history up to batch t. Define $S_{k,t}$ and $F_{k,t}$

as the number of successes and failures, respectively, observed for arm k up to batch t:

$$S_{k,t} = \sum_{(i,s):\, s \le t} \mathbb{1}\{A_{i,s} = k,\, Y_{i,s} = 1\}, \quad F_{k,t} = \sum_{(i,s):\, s \le t} \mathbb{1}\{A_{i,s} = k,\, Y_{i,s} = 0\}.$$

We adopt independent Beta$(1,1)$ priors (uniform on $[0,1]$) for the θ_k. By conjugacy, the posterior after batch t is

$$\theta_k \mid \mathcal{H}_t \sim \text{Beta}(\alpha_{k,t}, \beta_{k,t}), \qquad \alpha_{k,t} = S_{k,t} + 1,\ \beta_{k,t} = F_{k,t} + 1.$$

Thompson sampling proceeds as follows for batch $t+1$:

1. For each arm k, draw a single posterior sample

   $$\theta^*_{k,t} \sim \text{Beta}(\alpha_{k,t}, \beta_{k,t}),$$

   independently across arms.

2. Assign each unit in batch $t+1$ with probability $p_{k,t}$ for arm k

   $$p_{k,t} = \Pr(A_{t+1} = k \mid \mathcal{H}_t) = \Pr\left(\theta^*_{k,t} = \max_{j=1,\dots,K} \theta^*_{j,t} \,\middle|\, \mathcal{H}_t\right).$$

In practice, this assignment probability is not computed via the corresponding multidimensional integral; drawing one sample $\theta^*_{k,t}$ per arm and choosing the arm with the largest draw yields an exact Monte Carlo sample from the distribution $(p_{1,t}, \dots, p_{K,t})$. For alternative approaches, see Scott (2010).

Starting from the uniform prior Beta$(1,1)$, each observed success increases $\alpha_{k,t}$ by one and each failure increases $\beta_{k,t}$ by one. As $S_{k,t} + F_{k,t}$ grows, the posterior becomes more concentrated around the true θ_k, and the assignment probabilities $p_{k,t}$ converge toward playing the empirically best arm with a high probability. Figure 2 shows how the posterior distributions for a given arm become more peaked with additional batches.

2.2 Data structure of adaptive experiments

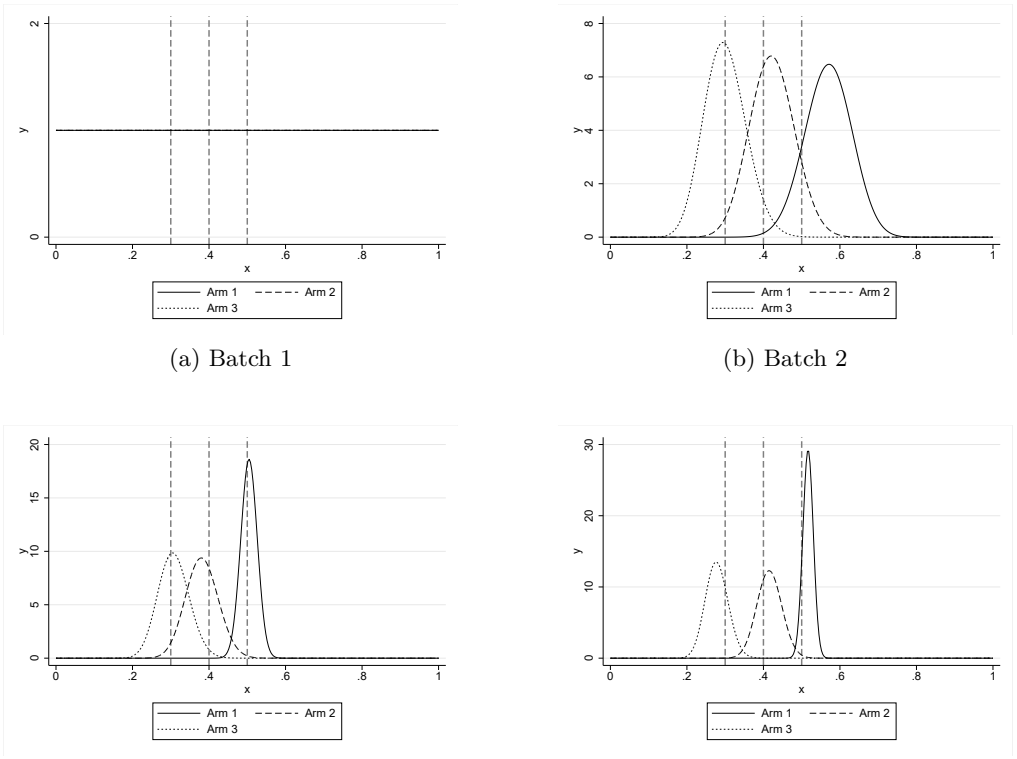
We consider batched adaptive experiments with periods $t = 1, \dots, T$. In each period, $N_{1,t}$ treated and $N_{0,t}$ control units are assigned based on past outcomes, and we observe outcomes $Y_{i,k,t}$ for arm $k \in \{0,1\}$. Within each period t, outcomes satisfy

$$Y_{i,k,t} = m_{k,t} + \varepsilon_{i,k,t}, \qquad E[\varepsilon_{i,k,t}] = 0,$$

with independent observations across individuals and possibly arm-specific variances $\sigma^2_{1,t}$ and $\sigma^2_{0,t}$; see Kemper and Rostam-Afschar (2025).

Treatment assignment across periods is adaptive (e.g., via a bandit algorithm), so the probability of receiving treatment depends on the realized history.

Figure 2: Posterior evolution under Beta–Bernoulli Thompson sampling



(a) Batch 1

(b) Batch 2

(c) Batch 5

(d) Batch 10

Notes: Thompson sampling with Bernoulli rewards and true success probabilities of 0.5, 0.4, and 0.3 for arms 1–3, respectively. The vertical line indicates the true success probability of each arm. The figure was generated using `bbandits_sim 0.5 0.4 0.3, size(200) batch(10) clipping(0.1) thompson plot_thompson`.

In each period t we summarize the causal effect by the difference in sample means

$$\hat{\Delta}_t \;=\; \bar{Y}_{1,t} - \bar{Y}_{0,t},$$

equivalently obtained as the coefficient on a treatment indicator from an OLS regression of $Y_{i,t}$ on the indicator within period t. Because sampling is i.i.d. *within* each period, standard OLS properties apply to $\hat{\Delta}_t$.

Table 1 illustrates a stylized data structure with two arms, here labeled A and B, corresponding to arms 1 and 0. The columns "Batch-Wise OLS" report $\hat{\Delta}_t$ for each batch, while the last column shows weights that are proportional to the period-specific precision used in the BOLS estimator.

The adaptive assignment of treatments makes naive pooling across periods prob-

Table 1: Stylized data structure. Here arm 1 corresponds to A and arm 0 to B.

Obs	Selected arm	Batch	Outcome	True Expected Outcome	OLS	Batch-Wise OLS	ω_t
1	A	0	0	0.5	0.600	0.500	$\sqrt{\frac{2\times2}{2+2}}$
2	B	0	0	0.2	0.167	0.000	$\sqrt{\frac{2\times2}{2+2}}$
3	A	0	1	0.5	0.600	0.500	$\sqrt{\frac{2\times2}{2+2}}$
4	B	0	0	0.2	0.167	0.000	$\sqrt{\frac{2\times2}{2+2}}$
5	A	1	0	0.5	0.600	0.500	$\sqrt{\frac{2\times2}{2+2}}$
6	B	1	1	0.2	0.167	0.500	$\sqrt{\frac{2\times2}{2+2}}$
7	A	1	1	0.5	0.600	0.500	$\sqrt{\frac{2\times2}{2+2}}$
8	B	1	0	0.2	0.167	0.500	$\sqrt{\frac{2\times2}{2+2}}$
9	A	2	0	0.5	0.600	0.667	$\sqrt{\frac{1\times3}{1+3}}$
10	A	2	1	0.2	0.600	0.667	$\sqrt{\frac{1\times3}{1+3}}$
11	A	2	1	0.5	0.600	0.667	$\sqrt{\frac{1\times3}{1+3}}$
12	B	2	0	0.2	0.167	0.000	$\sqrt{\frac{1\times3}{1+3}}$
13	A	3	1	0.5	0.600	0.667	$\sqrt{\frac{1\times3}{1+3}}$
14	A	3	0	0.2	0.600	0.667	$\sqrt{\frac{1\times3}{1+3}}$
15	A	3	1	0.5	0.600	0.667	$\sqrt{\frac{1\times3}{1+3}}$
16	B	3	0	0.2	0.167	0.000	$\sqrt{\frac{1\times3}{1+3}}$
OLS					$\widehat{\text{Outcome}} = 0.6 - 0.433 \times \mathbb{1}_{\text{arm B}}$		
BOLS					$-0.443 = \frac{\sqrt{\frac{2\times2}{2+2}}\times0.5+\sqrt{\frac{2\times2}{2+2}}\times0+\sqrt{\frac{1\times3}{1+3}}\times0.667+\sqrt{\frac{1\times3}{1+3}}\times0.667}{\sqrt{\frac{2\times2}{2+2}}+\sqrt{\frac{2\times2}{2+2}}+\sqrt{\frac{1\times3}{1+3}}+\sqrt{\frac{1\times3}{1+3}}}$		
					$\widehat{\text{Outcome}} = 0.6 - 0.443 \times \mathbb{1}_{\text{arm B}}$		

Note: Fictional data for a batched adaptive experiment with two arms A and B. Batch 0 is randomized; later batches are assigned adaptively (e.g. by Thompson sampling). "Batch-Wise OLS" reports $\hat{\Delta}_t$ and ω_t are weights proportional to period-specific precision under homoskedasticity.

lematic: standard OLS that ignores that endogenous treatment assignment may lead to asymptotic non-normality and invalid tests (Zhang et al. 2020). To address this, Kemper and Rostam-Afschar (2025) propose a heteroskedasticity-robust batched OLS (BOLS) estimator that aggregates the $\hat{\Delta}_t$ using precision weights.

For each period t, the variance of $\hat{\Delta}_t$ is

$$v_t = \frac{\sigma^2_{1,t}}{N_{1,t}} + \frac{\sigma^2_{0,t}}{N_{0,t}}.$$

Define the period weights $w_t = 1/\sqrt{v_t}$ and $S = \sum_{s=1}^{T} w_s$. The BOLS estimator is the

precision-weighted average

$$\hat{\Delta} = \sum_{t=1}^{T} \frac{w_t}{S}\, \hat{\Delta}_t.$$

Intuitively, periods that are larger, more balanced, and less noisy receive more weight. Under homoskedasticity, w_t is proportional to $\sqrt{N_{1,t}N_{0,t}/(N_{1,t}+N_{0,t})}$, which is the quantity shown as ω_t in Table 1.

3 Inference about causal effects

Following Zhang et al. (2020), Kemper and Rostam-Afschar (2025) show that the combined estimator $\hat{\Delta}$ is asymptotically normal under large-batch asymptotics in adaptive experiments. By construction,

$$\mathrm{Var}(\hat{\Delta}) = \frac{T}{S^2}, \qquad \mathrm{SE}(\hat{\Delta}) = \frac{\sqrt{T}}{S}.$$

In practice, v_t and w_t are replaced by consistent estimates $\hat{v}_t$ and $\hat{w}_t = 1/\sqrt{\hat{v}_t}$, and $\hat{S} = \sum_t \hat{w}_t$.

For testing $H_0 : \Delta = c$, the feasible BOLS test statistic is a combination of period z-scores $z_{t,\mathrm{het}} = \frac{\hat{\Delta}_t - c}{\sqrt{v_t}}$. In other words, it is the difference of the margin to the test value $\hat{\Delta} - c$ weighted with its aggregate standard error $\widehat{\mathrm{SE}}(\hat{\Delta}) = \frac{\sqrt{T}}{\hat{S}}$:

$$\hat{Z}_{\mathrm{het}} = \frac{\hat{\Delta} - c}{\widehat{\mathrm{SE}}(\hat{\Delta})} = \frac{1}{\sqrt{T}} \sum_{t=1}^{T} z_{t,\mathrm{het}} = \frac{1}{\sqrt{T}} \sum_{t=1}^{T} \frac{\hat{\Delta}_t - c}{\sqrt{v_t}},$$

which is asymptotically standard normal. Hence a two-sided $(1-\alpha)$ confidence interval for Δ is

$$CI_{1-\alpha} = \hat{\Delta} \pm z_{1-\alpha/2}\, \widehat{\mathrm{SE}}(\hat{\Delta}),$$

which in applications can be well approximated by

$$CI \approx \hat{\Delta} \pm 1.96 \cdot \frac{\sqrt{T}}{\hat{S}}.$$

4 Properties of estimators for batched bandits: Evidence from Monte Carlo simulations

The Monte Carlo simulations in Tables 2 and 3 evaluate the finite-sample behavior of OLS and heteroskedasticity-robust BOLS estimators for the batched difference in sample means, $\hat{\Delta}_t = \bar{Y}_{1,t} - \bar{Y}_{0,t}$, under adaptive treatment assignment. Three main findings emerge that provide clear guidance for the design of batched bandit experiments.

First, the simulations show that *batch size is the key determinant* of valid inference. Table 2 varies the number of observations per batch and demonstrates that standard

OLS exhibits substantial size distortions when batches are small. For example, when $N_t = 5$ and the true effect $\Delta = 0$, OLS rejects the null hypothesis almost 8% of the time rather than the nominal 5%, see Table 2, $\Delta = 0.0$, $N_t = 5$. BOLS remains markedly closer to correct size under the same conditions. As batch sizes increase, both estimators stabilize, but BOLS reaches nominal size considerably earlier. With $N_t \approx 20$, BOLS performs well even in small-margin settings, consistent with the theoretical large-batch-size asymptotics of Kemper and Rostam-Afschar (2025).

Table 2: Bernoulli Thompson Sampling - Small batch size

		OLS/Batch				BOLS/Batch			
N_t	$\Delta[Y_{t,k}]$	10	25	50	100	10	25	50	100
5	0.0	0.001	0.001	0.000	0.000	0.000	0.001	0.000	0.000
		(0.076)	(0.067)	(0.060)	(0.058)	(0.062)	(0.057)	(0.052)	(0.051)
5	0.1	0.105	0.107	0.107	0.105	0.101	0.102	0.102	0.101
		(0.075)	(0.063)	(0.055)	(0.048)	(0.058)	(0.056)	(0.047)	(0.049)
5	0.2	0.213	0.205	0.205	0.202	0.205	0.200	0.202	0.201
		(0.066)	(0.056)	(0.051)	(0.051)	(0.056)	(0.052)	(0.052)	(0.050)
10	0.0	-0.001	0.000	-0.000	0.000	-0.001	0.001	-0.000	0.000
		(0.066)	(0.060)	(0.054)	(0.055)	(0.058)	(0.053)	(0.049)	(0.050)
10	0.1	0.106	0.105	0.104	0.103	0.102	0.102	0.101	0.101
		(0.060)	(0.056)	(0.050)	(0.049)	(0.051)	(0.051)	(0.050)	(0.052)
10	0.2	0.208	0.204	0.201	0.201	0.204	0.201	0.200	0.200
		(0.059)	(0.048)	(0.051)	(0.047)	(0.053)	(0.049)	(0.051)	(0.045)
20	0.0	0.001	0.001	0.000	-0.000	0.001	0.001	0.000	-0.000
		(0.058)	(0.060)	(0.055)	(0.055)	(0.052)	(0.053)	(0.050)	(0.050)
20	0.1	0.105	0.104	0.102	0.100	0.102	0.102	0.101	0.100
		(0.056)	(0.056)	(0.048)	(0.046)	(0.048)	(0.053)	(0.051)	(0.050)
20	0.2	0.203	0.202	0.201	0.200	0.200	0.201	0.201	0.200
		(0.049)	(0.049)	(0.053)	(0.049)	(0.051)	(0.049)	(0.054)	(0.049)
100	0.0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
		(0.056)	(0.058)	(0.051)	(0.058)	(0.048)	(0.051)	(0.046)	(0.054)
100	0.1	0.102	0.100	0.100	0.100	0.101	0.100	0.100	0.100
		(0.050)	(0.053)	(0.048)	(0.048)	(0.053)	(0.053)	(0.047)	(0.049)
100	0.2	0.200	0.200	0.200	0.200	0.200	0.200	0.200	0.200
		(0.052)	(0.053)	(0.046)	(0.052)	(0.049)	(0.053)	(0.047)	(0.051)

Note: The Monte Carlo simulation is based on 10,000 iterations. Type-I error (α) for the 95-percentile in parentheses. The true 95-percentile of the standard normal distribution implies a rejection rate of 0.05. $\Delta[Y_{t,k}]$ is the true margin between the two arms.

Second, the simulations underscore the importance of *heteroskedastic precision weighting* when treatment assignment probabilities evolve adaptively. When the true effects are small, assignment probabilities do not concentrate, and the variance of the pooled OLS estimator depends on stochastic treatment shares, producing systematic overrejection. This pattern is particularly visible in the $\Delta = 0$ and $\Delta = 0.1$ rows of Table 2. In contrast, BOLS aggregates the batchwise estimates $\hat{\Delta}_t$ using estimated variances

that remain valid even with small margins. Point estimates from OLS and BOLS are nearly identical across all designs, but their inferential properties differ sharply. For this reason, BOLS should be used for hypothesis testing.

Table 3: Bernoulli Thompson Sampling - Few batches

		OLS/Batch				BOLS/Batch			
N_t	$\Delta[Y_{t,k}]$	1	3	5	8	1	3	5	8
100	0.0	0.001	-0.000	-0.000	0.000	0.001	-0.001	-0.000	0.000
		(0.056)	(0.057)	(0.054)	(0.055)	(0.055)	(0.053)	(0.051)	(0.052)
100	0.1	0.100	0.102	0.102	0.102	0.100	0.101	0.101	0.100
		(0.054)	(0.053)	(0.052)	(0.051)	(0.052)	(0.052)	(0.051)	(0.053)
100	0.2	0.199	0.202	0.2	0.199	0.199	0.202	0.200	0.199
		(0.056)	(0.050)	(0.053)	(0.048)	(0.053)	(0.050)	(0.054)	(0.049)
200	0.0	0.001	0.000	0.000	-0.000	0.001	0.001	0.000	-0.000
		(0.055)	(0.052)	(0.055)	(0.057)	(0.054)	(0.050)	(0.052)	(0.050)
200	0.1	0.100	0.101	0.102	0.101	0.100	0.101	0.101	0.100
		(0.052)	(0.052)	(0.045)	(0.048)	(0.051)	(0.054)	(0.049)	(0.052)
200	0.2	0.200	0.200	0.200	0.200	0.200	0.199	0.200	0.200
		(0.050)	(0.048)	(0.053)	(0.052)	(0.049)	(0.050)	(0.051)	(0.053)
500	0.0	-0.000	0.000	-0.000	0.000	-0.000	-0.000	-0.000	-0.000
		(0.059)	(0.052)	(0.054)	(0.054)	(0.058)	(0.050)	(0.051)	(0.050)
500	0.1	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100
		(0.050)	(0.048)	(0.048)	(0.050)	(0.049)	(0.050)	(0.049)	(0.052)
500	0.2	0.200	0.200	0.200	0.200	0.200	0.200	0.200	0.200
		(0.053)	(0.049)	(0.050)	(0.050)	(0.053)	(0.049)	(0.049)	(0.048)
1000	0.0	-0.000	0.000	0.000	-0.000	-0.000	0.000	0.000	-0.000
		(0.049)	(0.056)	(0.050)	(0.057)	(0.049)	(0.053)	(0.048)	(0.051)
1000	0.1	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100
		(0.046)	(0.049)	(0.048)	(0.048)	(0.046)	(0.047)	(0.049)	(0.048)
1000	0.2	0.200	0.200	0.200	0.200	0.200	0.200	0.200	0.200
		(0.049)	(0.051)	(0.050)	(0.047)	(0.048)	(0.051)	(0.052)	(0.048)

Note: The Monte Carlo simulation is based on 10,000 iterations. Type-I error (α) for the 95-percentile in parentheses. The true 95-percentile of the standard normal distribution implies a rejection rate of 0.05. $\Delta[Y_{t,k}]$ is the true margin between the two arms. The clipping rate is 0.2.

Third, Table 3 shows that the *number of batches matters much less than batch size*. When batch sizes are sufficiently large ($N_t \geq 100$), both OLS and BOLS achieve correct size even with as few as one or three batches. Increasing the number of batches from 1 to 8 yields only minor improvements, whereas increasing the batch size has a substantial effect. This is consistent with the theoretical results for BOLS, which rely primarily on large-N_t asymptotics rather than large-T asymptotics.

Taken together, the evidence suggests clear practical advice. Researchers should prioritize adequately large batch sizes to ensure unbiased estimation of Δ and valid inference for adaptively collected data. While both OLS and BOLS produce similar point estimates of the treatment effect, only BOLS consistently delivers correct size

and coverage, especially when treatment effects are small. Our results also show that OLS gives correct coverage when margins are large. For applied work, we therefore recommend reporting both estimators but relying on BOLS confidence intervals for inference.

5 The `bbandits` commands

The `bbandits` package provides four commands to analyze, simulate, and conduct adaptive experiments with batched bandits. i) `bbandits`, ii) `bbandits_sim`, iii) `bbandits_initialize`, iv) `bbandits_update`.

The `bbandits` command relies on a Python backend. On first use, the required Python environment is initialized, which may take a few moments. Subsequent calls typically execute without delay.

Before using the package, users must install Python (version 2.7 or higher) and the following Python packages: `scikit-learn`, `numpy`, `pandas`, `scipy`, `statsmodels`, and `sfi`. Installation instructions for Python and its packages are available at https://www.python.org/. An additional step-by-step installation guide for using Python with Stata is provided by Achim Ahrens, Christian Hansen, Mark Schaffer, and Thomas Wiemann. Ahrens et al. (2022)

Once Python is installed, the `bbandits` package can be installed in Stata via

```
. net from https://rostam-afschar.de/bbandits
. net install bbandits.pkg
```

5.1 Analyzing bbandit data

The **bbandits** command provides the BOLS and OLS estimates with correct confidence intervals. Additionally, it provides a graphical analysis.

Syntax

Batched bandit analysis can be implemented in Stata using the following command syntax:

`bbandits` *reward chosen_arm batch* [`, reference_arm(integer) test_value(#)`
`plot_thompson stacked no_plot twooptions_thompson(string)`
`twooptions_bols(string) twooptions_ols(string)`
`twooptions_sharebybatch(string) twoptions_stackedsharebybatch(string)`
`twooptions_cumsharesbyybatch(string)`]

The **bbandits** syntax requires three variables. It takes the dependent variable *re-*

ward a measure of the outcome or regret, typically of success or failure. The second input requires a categorical variable, *chosen_arm*, that indicates which of the k arms was assigned to the unit. As a third input, **bbandits** takes a categorical variable, which indicates in which batch t of T batches the reward was observed.

Options

`reference_arm(integer)` specifies which arm is taken as a reference arm (control group). The treatment effects are then calculated with respect to the reference arm. The default reference arm is arm 0 (first of K arms).

`test_value(#)` specifies the null-hypothesis. The default value is 0.

`plot_thompson` plots beta distributions for each of the treatments.

`stacked` plots beta distributions for each of the treatments but vertically stacked.

`no_plot` specifies that only the output table without any plots is displayed.

`twoptions_thompson(string)` takes user-specific two-way options for the two-way Thompson plot.

`twoptions_bols(string)` takes user-specific two-way options for the plot of the BOLS treatment effects.

`twoptions_ols(string)` takes user-specific two-way options for the plot of the BOLS and the OLS treatment effects.

`twoptions_sharebybatch(string)` takes user-specific two-way options for the plot of the shares assigned to each treatment arm by batch.

`twoptions_stackedsharebybatch(string)` takes user-specific two-way options for the plot of the shares assigned to each treatment arm by batch but stacked as an area.

`twoptions_cumsharesbyybatch(string)` takes user-specific two-way options for the plot of the cumulative shares assigned to each treatment arm by batch stacked as an area.

Returned results

Scalars

`e(N)`	number of observations

Matrices

`e(res)`	matrix with all output results
`e(batch_ols_coefficients)`	matrix with all batched OLS coefficients
`e(batched_ols_weights)`	matrix containing the BOLS weights
`e(reward_evaluation)`	Matrix comparing rewards in adaptive and classic experiments

The matrix $e(res)$ includes for each arm the OLS margins, the BOLS margins, the z statistics, the p-values, the BOLS 95% confidence intervals, the observations of the reference arm, the observations of the treatment arm, the heteroskedasticity-

robust BOLS standard errors, the treatment arm indicator and the OLS 95% confidence intervals. In addition, all returned results from **regress** are returned.

5.2 Simulate bandit experiments

Syntax

The following command can simulate a bandit experiment to explore the data structure and mechanism of the algorithm. Additionally, it allows the user to conduct a simple Monte Carlo study to examine the inference in different scenarios.

```
bbandits_sim true_expected_values [ , batch(integer) size(integer) eps(#)
    decay(#) clipping(#) exploration_phase(integer) greedy thompson
    plot_thompson twopts(string) stacked monte_carlo n(integer)
    reference_arm(#) arm(integer) standard_deviations(numlist) ]
```

The **bbandits_sim** syntax requires the scalar parameters *true_expected_values*, for example, success parameters 0.2 0.3 0.4, to specify the reward distributions. For the ε-greedy algorithm rewards are drawn from a normal distribution with the specified expected value and a standard deviation of 1. For Thompson Sampling, the rewards are drawn from a Bernoulli distribution. Two arms are required but there could be more. A series of options are allowed to further calibrate the algorithm. The two popular classes of bandit algorithms, ε-greedy and Thompson sampling, can be chosen in the options. The default algorithm is ε-greedy. The default case is to simulate a single bandit experiment and the return of the simulated data set. Users can also run a simple Monte Carlo study which is specified by the respective option.

Options

`batch(integer)` specifies the number of batches. The default is 25 batches.

`size(integer)` specifies the batch size. The default is 100 observations per batch.

`eps(#)` specifies the ε parameter for the ε-greedy algorithm. The default ε is 10%.

`decay(#)` is a linear decay rate for the ε-greedy algorithm. The default value is 1.

`clipping(#)` specifies the clipping rate for the Bernoulli Thompson algorithm. The default value is 5%.

`exploration_phase(integer)` specifies the number of batches where the algorithm assigns the treatment uniformly. The default value is 0.

`greedy` specifies the ε-greedy algorithm instead of the Bernoulli-Thompson algorithm. The default is ε-greedy.

`standard_deviations(numlist)` specifies the standard deviations as parameters of the

rewards of the normal distribution for the ε-greedy simulated data.

thompson specifies the Bernoulli-Thompson Sampling algorithm instead of the ε-greedy treatment assignment algorithm.

plot_thompson generates plots of the beta distribution under the Thompson sampling algorithm.

stacked plots the beta distribution under the Thompson sampling algorithm but vertically stacked.

monte_carlo specifies a Monte Carlo simulation with 1,000 repetitions. Only two arms can be compared to each other even if multiple arms are specified.

n(integer) specifies the number of repetitions for the Monte Carlo simulation.

reference_arm(integer) specifies the reference arm for the Monte Carlo simulation.

arm(integer) specifies the arm which is compared against the reference arm in the Monte Carlo simulation. The Monte Carlo simulation function only allows to compute the test statistic for two arms but the data can be drawn from a multi-arm simulation.

twopts(string) takes user-specific two-way options.

Return

The simulation returns a simulated data set. The Monte Carlo simulation returns the test statistics for the OLS and BOLS. Additionally, the stored result e(decay_rate) is returned. It captures the ε or clipping rate for each batch and should become smaller when a decay rate is determined.

Matrices

e(decay_rate)	ε or clipping rate for each batch.

5.3 Conducting bandit experiments interactively

To easily conduct a bandit experiment, our package provides two commands. Below, in section 7, we provide an example of how to use these commands. First, the **bbandits_initialize** command helps to set up a compatible data structure for an adaptive experiment and the package. Second, the **bbandits_update** function implements the bandit algorithms and returns which treatment arms to assign for the next batch based on the chosen bandit algorithm. Both commands are briefly explained below.

Syntax

```
bbandits_initialize [ , batch(integer) arms(integer)
    exploration_phase(integer) sae ]
```

For **bbandits_initialize** the user must load a data set where each row is one of the potentially treated units.

Options

batch(integer) specifies the number of batches. The default number is 3.

arms(integer) specifies the number of treatment arms. The default is 2.

exploration_phase(integer) specifies the number of batches where the algorithm assigns the treatment uniformly. The default value is 1.

sae specifies initialization according to the arm-elimination algorithm described in Esfandiari et al. (2021), see section 2.1.

Return

This command generates the following three new variables. First, the variable *reward* is generated but contains missings, because the experiment has not yet started. The user must fill this variable later with the experimental data. Second, the variable *chosen_arm* is generated. This variable contains the assigned treatment arms. For *exploration_phase*, treatment arms are assigned uniformly and during the adaptive portion of the experiment algorithmically. Third, the variable *batch* indicates the respective batch. **bbandits_initialize** generates batches with equal batch sizes.

Syntax

The **bbandits_update** command has the following syntax:

```
bbandits_update [ , thompson greedy sae clipping(#) epsilon(#)
   active_arms(numlist) batch_sae(#) excel("path") ]
```

During the adaptive portion of the experiment, the bandit algorithm chosen by the user assigns the treatment for the next batch. After having carried out the treatment, the user records the *reward*. The user then enters the newly observed rewards into the data set and runs **bbandits_update** again to start the next iteration of the algorithm that assigns units again to treatments for the next batch based on the history of rewards. Once all units are treated and the rewards are recorded, the **bbandits** command can be used to analyze the data.

Options

thompson specifies Bernoulli Thompson sampling algorithm.

greedy specifies the ε-greedy algorithm.

sae specifies updating according to the Successive Arm Elimination (SAE) algorithm in section 2.1.

clipping(#) specifies the clipping rate for the Bernoulli Thompson algorithm. The default value is 0.05.

epsilon(#) specifies the ε rate for the ε-greedy algorithm. The default value is 0.1.

active_arms(numlist) is a *numlist* indicating the active arms according to the successive arm-elimination algorithm.

batch_sae(#) requires the total number of batches in the successive arm elimination setting.

excel("path") indicates that the updated data is saved as an Excel file under the specified path. The saved file can be used to impute the newly observed rewards.

Return

The **bbandits_update** command returns a data set that contains the treatment assignments for the next batch according to the selected bandit algorithm. It also generates a variable *chosen_arm_numeric*, a numeric code that identifies the assigned treatment arm for each observation.

Scalars

e(current_batch)	Returns the current batch.

Matrices

e(arm_labels)	Numeric label
e(probabilities)	Share that the corresponding arm should be played in the next batch.

The returned matrix *e(arm_labels)* is a numeric label that corresponds to the numeric label for the assigned arm from the newly generated variable *chosen_arm_numeric*. The order of e(probabilities). corresponds to the *e(arm_labels)*. So, the first value is the share of arm 0, the second is the share of arm 1, and so on.

6 Examples with empirical application

In this section, we present applications of the **bbandits** command to analyse and conduct inference about effect sizes ex-post using data from adaptive experiments. The applications differ mainly in terms of treatment assignment algorithm and the number of treatment arms.

The first application is taken from Kasy and Sautmann (2021) and uses exploration sampling (explained below) for six treatment arms. The second application is based on Gaul et al. (2025) and uses Thompson sampling for 32 treatment arms.

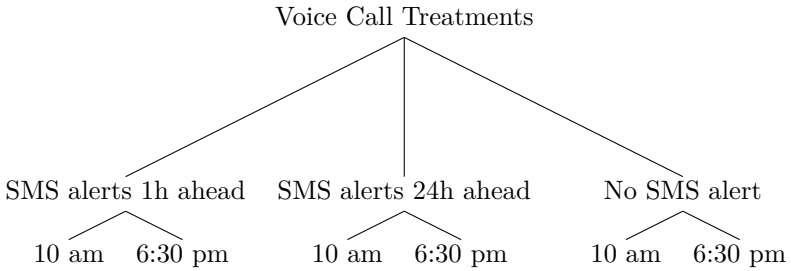
6.1 Six call methods to enroll rice farmers (Kasy and Sautmann 2021)

Kasy and Sautmann (2021) designed an experiment using *exploration sampling* (see below and Section 4 in their paper) to help Precision Agriculture for Development (PAD) choose among a variety of different call methods to enroll rice farmers in one state in India. PAD is an NGO that works with government partners to provide a phone-based personalized agricultural extension service to farmers.

The outcome (reward) is a binary variable describing call completion:

$$\text{call completed} = \begin{cases} 1 & \text{if call recipient answered five questions asked during call,} \\ 0 & \text{otherwise.} \end{cases}$$

PAD tested six automated voice calls treatments. Call time varied to be either in the morning (10 am) or in the evening (6:30 pm). The calls were made with SMS text message alerts sent 1 hour ahead, 24 hours ahead, or without SMS alerts.



Kasy and Sautmann (2021) base their algorithm on Thompson sampling and propose a modified treatment assignment algorithm which they call *exploration sampling*. In Thompson sampling, the probability that arm k is best in terms of reward at time t $p_t^k = P(E_\theta[R_k, t] = \max\{E_\theta[R_1], \ldots, E_\theta[R_K]\}|R_t)$ can be used for treatment assignment. *Exploration sampling* replaces the Thompson assignment shares

$$(p_t^1, p_t^2, ..., p_t^k)$$

with the following transformed shares $q_t^k = \frac{p_t^k(1-p_t^k)}{\sum_k p_t^k(1-p_t^k)}$.

This modification shifts weight away from the best performing option to its close competing treatments. Since there is at most one k for which $p_t^k > 1/2$, q_t^k monotonically increases and is concave in p_t^k.

A list of 10,000 valid phone numbers were randomly assigned to one of 16 batches with 600 numbers each (and one with 400). Starting on June 3, 2019, a new experimental wave was started every other day and completed the next day.

The success rate of each treatment arm was estimated starting with a uniform prior in order to determine the assignment frequencies for each consecutive wave using exploration sampling.

```
. use "example data\kasy_sautmann_2021.dta", clear
. bbandits outcome treatment date
BOLS calculation for arm: 1
BOLS calculation for arm: 2
BOLS calculation for arm: 3
BOLS calculation for arm: 4
BOLS calculation for arm: 5
```

N	=	10000				
Best-arm total	=	1926	Best-arm mean	=	0.1926	
Actual total	=	1804	Actual mean	=	0.1804	
Uniform total	=	1709	Uniform mean	=	0.1709	

Share b	Mean reward arm b
0.0903	0.1606

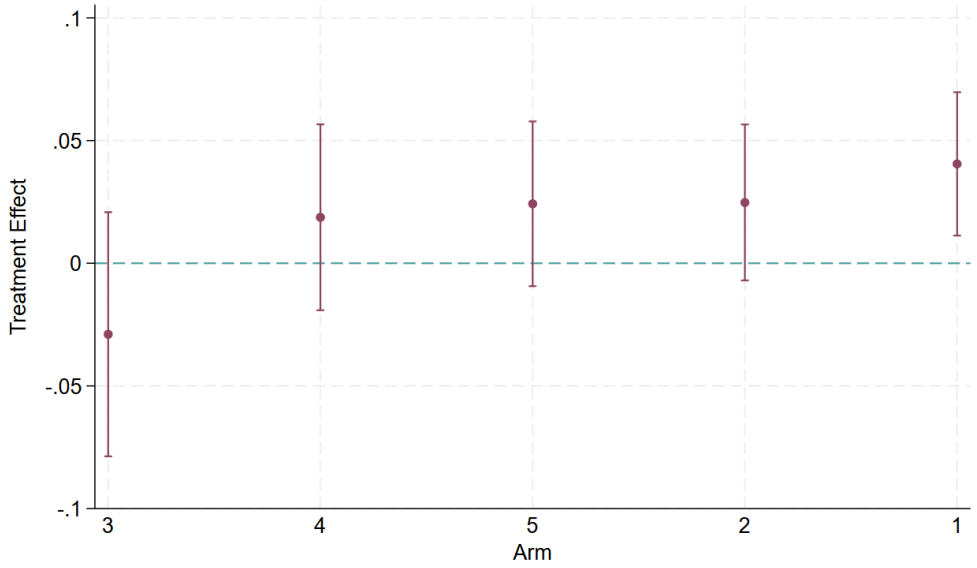
Share k	Arm/Est.	Marg.	Robust SE	z	P>|z|	[95% CI]
0.3931	1-0					
	OLS	0.0320	0.0137	2.33	0.020	[0.0051, 0.0589]
	BOLS	0.0405	0.0149	2.72	0.007	[0.0113, 0.0697]
0.2234	2-0					
	OLS	0.0185	0.0147	1.26	0.208	[-0.0103, 0.0472]
	BOLS	0.0248	0.0162	1.53	0.127	[-0.0070, 0.0565]
0.0366	3-0					
	OLS	-0.0158	0.0221	-0.71	0.475	[-0.0591, 0.0275]
	BOLS	-0.0289	0.0254	-1.14	0.254	[-0.0787, 0.0208]
0.1081	4-0					
	OLS	0.0078	0.0167	0.47	0.641	[-0.0250, 0.0405]
	BOLS	0.0187	0.0193	0.97	0.332	[-0.0191, 0.0566]
0.1485	5-0					
	OLS	0.0192	0.0158	1.22	0.223	[-0.0117, 0.0501]
	BOLS	0.0242	0.0171	1.41	0.157	[-0.0094, 0.0578]

Cumulatively, 39.31% of farmers received the most successful type of call. The least successful call (at 6:30 pm without a text message alert) was received by only 3.66%. Based on the posterior estimated success rates, exploration sampling not only improved learning, but also increased overall success rates within the experiment (18.04%) compared to a standard design with equal assignment to treatment arms (where the estimated success rate based on posterior means would be 17.09%).

The OLS results can be interpreted as follows: With over 75% probability, calling farmers at 10 am after a text message an hour ahead of time has the greatest success rate, estimated to be 19.3% (16.06%+3.20%). Across treatments, higher success rates are associated with a higher number of observations, and correspondingly smaller posterior standard deviation. Both point estimates and standard errors from BOLS are very similar to those obtained with OLS. Figure 3 shows the treatment margins for each treatment compared to the reference treatment (10 am without SMS alert).

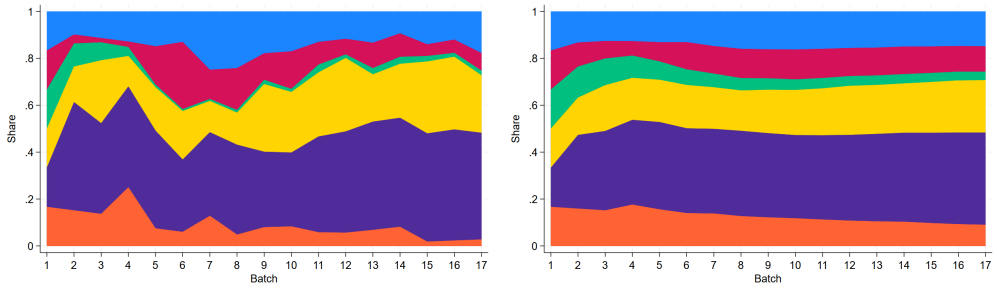
The Figures 4a and 4b show that one treatment was assigned to the most participants from wave 2 onwards, but some closely competing treatments received a high share of observations, especially in early waves. The number of observations per wave assigned to each of the treatments stabilized in later waves.

Figure 3: BOLS Treatment effect for the six call methods to enroll rice farmers



Notes: The Figure shows the BOLS estimates of margins relative to the control arm with their respective asymptotic 95% confidence intervals. The estimated margin for arm 1 is 3.20% (OLS) and 4.06% (BOLS). The figure was generated using `kasy_sautmann_2021.dta` and running `bbandits outcome treatment date`.

Figure 4: Treatment assignment of the six call methods to enroll rice farmers over batches



(a) Batchwise shares (b) Cumulative shares

Notes: This figure shows the shares assigned over the experiment to each of the 32 arms in each batch and cumulatively by batch. The treatment assignment algorithm is Exploration sampling. Arm 1 is selected most frequently with 39.31% overall. The cumulative shares show a quite balanced exploration. The figure was generated using `kasy_sautmann_2021.dta` and running `bbandits outcome treatment date`.

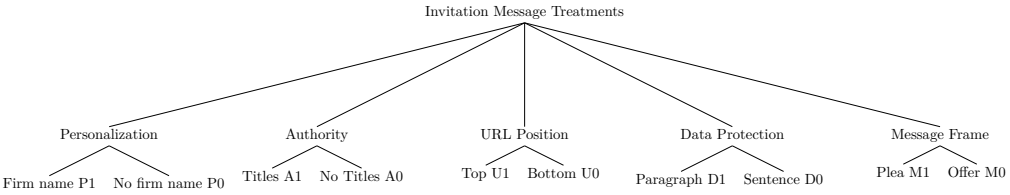
6.2 32 invitation messages for business surveys (Gaul et al. 2025)

Gaul et al. (2025) designed an experiment using *Thompson sampling* (see Section 2.1 and Section 3 in their paper) to support the German Business Panel (GBP) in selecting among a variety of different invitation messages to survey firm decision makers in Germany. The GBP is a web-based survey of firm decision makers in Germany that distributes email invitations on each working day (see Bischof et al. 2025, Hack and Rostam-Afschar 2024 for details).

The outcome (reward) is a binary variable describing the start of the survey:

$$\text{survey started} = \begin{cases} 1 & \text{if the email invitation recipient started the survey,} \\ 0 & \text{otherwise.} \end{cases}$$

The GBP tested five components of invitation letters and their full interactions in $2^5 = 32$ treatments. The first treatment varies personalization by mentioning or not mentioning the firm name, the second varies the authority of the sender by listing the official full academic titles along with the senders' names or their names only, the third varies the position of the URL to start the survey between being at the top or at the bottom of the invitation message, the fourth emphasizes data protection either in a separate paragraph with two strongly phrased sentences or in a single sentence, and the fifth varies the message frame by either pleading for support for the survey or simply offering to participate.



Gaul et al. (2025) apply the Thompson sampling algorithm and use an assignment rule based on the posterior probability that arm k is optimal in terms of reward at time t.

A total of 176,000 contacts from public and private firms in Germany entered the experiment and were assigned to 15 weekly batches between August 16, 2022 and November 25, 2022. During the first four batches, a fixed and balanced burn-in phase was implemented in which each treatment was assigned with probability 1/32. Starting from batch 5, success rates were estimated weekly and the assignment frequencies of the 32 treatments were updated according to the Thompson sampling rule for each subsequent batch.[1]

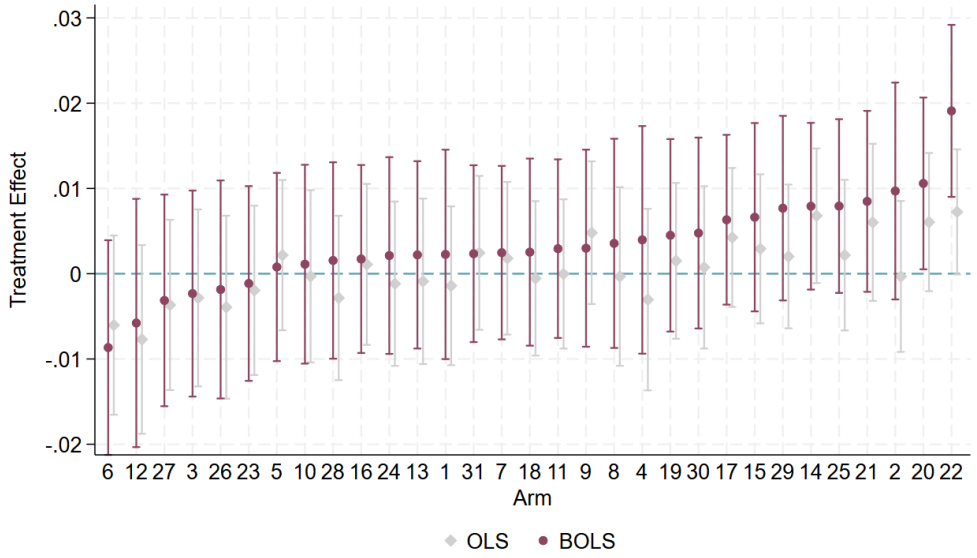
The results can be interpreted as follows: The invitation message combining personalization, high authority, a top position of the URL, no strong emphasis on data

1. In the burn-in phase, the treatment P0A0U0D1M0 performed particularly poorly and therefore received a very small assignment probability thereafter. In the main experiment, a clipping constraint ensured a strictly positive minimum assignment probability for each arm.

protection, and pleading for support yields the highest success rate (P1A1U1D0M1). The least successful invitation message is its exact opposite (P0A0U0D1M0). Across treatments, higher success rates are associated with a higher number of observations, and correspondingly narrower confidence intervals due to the adaptive allocation mechanism. Figure 5 shows the estimated margins relative to the reference arm P0A0U0D0M0.

Cumulatively, 18.40% of firm decision makers received the most successful invitation letter, while the least successful invitation letter (P0A0U0D1M0) was received by only about 1%. Based on the posterior estimated success rates and the realized assignment shares, Thompson sampling not only improved learning but also increased overall success within the experiment. Relative to a counterfactual design with equal assignment to all treatment arms, the realized adaptive design generated about 5-7% more survey starts.

Figure 5: BOLS and OLS Treatment effects for 32 invitation messages

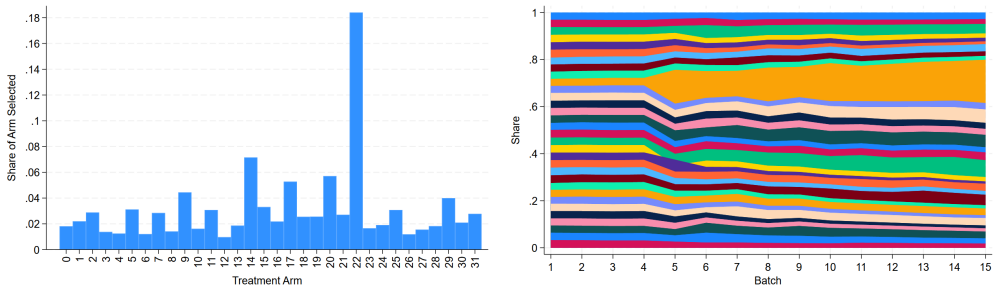


Notes: This figure shows BOLS estimates along with corresponding OLS estimated margins relative to the reference arm for each of the 32 arms. The treatment assignment algorithm is Thompson sampling. The figure illustrates that the best-performing arms exhibit both higher estimated starting rates and narrower confidence intervals due to higher effective sample sizes. The figure was generated using `gaul_et_al_2024.dta` and running `bbandits reward selected trial`.

Focusing on the five components instead of the full interaction effects of the invitation messages, personalization increases the starting rate significantly by 0.41 percentage points (p-value: <0.001). High authority and pleading for help also increase the starting rate by 0.16 (p-value: 0.039) and 0.25 percentage points (p-value: 0.004), respectively. Emphasizing data protection has a negative effect on the starting rate, while the position of the URL has no statistically significant effect. Given the overall

low baseline starting rate, these effects are also economically meaningful. For example, relative to the marginal mean with an unpersonalized invite of about 4%, personalizing the message increases the propensity to start answering the questionnaire by roughly 10%.

Figure 6: Treatment assignments for 32 invitation messages



(a) Total frequency of treatment assignment (b) Cumulative shares

Notes: This figure shows the shares assigned over the experiment to each of the 32 arms and the cumulative shares assigned to each arm by batch. The treatment assignment algorithm is Thompson sampling. The best-performing arm (P1A1U1D0M1) is selected most frequently with 18.40% overall. The cumulative shares show that this arm was selected more frequently immediately after the burn-in phase. The figure was generated using `gaul_et_al_2024.dta` and running `bbandits reward selected trial`.

Figures 6 shows that one treatment (P1A1U1D0M1) was assigned to the most participants from batch 5 onwards, while several closely competing treatments received substantial shares in the early waves. Thompson sampling increasingly allocated more observations per batch to the better performing arms over time.

7 Designing and conducting bandit experiments

A suggested workflow to run your own adaptive experiments with **bbandits** is as follows. First, you need to initialize the experiment: i) create a list of all potentially treated units, ii) separate them into batches using **bbandits_initialize**, iii) determine the number of batches for the exploration phase, and iv) decide on the number of treatments (in the exploration phase the treatment arms will automatically be randomly assigned). Next, v) enter the rewards from the first iteration, and vi) configure the updating algorithm with **bbandits_update**, specifying the `clipping` rate when using Thompson sampling or `epsilon` when using ε-greedy.

After initialization, i) assign treatments according to the weights provided by the algorithm, ii) enter the observed rewards and update the algorithm with **bbandits_update**, and iii) repeat this process until rewards are observed for all units. Finally, iv) analyze the adaptively generated data using **bbandits**.

7.1 Initialization

The **bbandits** package provides two functions to run batched bandits experiments. To illustrate the standard workflow think of a setting like in Duflo et al. (2012) where the outcome variable is the absence of a school teacher and the two treatment arms are financial incentives for attending class and video surveillance. A control arm receives no incentives. Instead of running a classic field experiment as Duflo and co-authors do, one could use the **bbandits** package to conduct an adaptive experiment.

To start the experiment, compiling a list of all participating units is recommended. In the illustrative example, these would be school teachers where some are control units while others receive one of the two treatment arms (surveillance or financial incentives). The rewards will be simulated here. Once the list of all participating subjects is loaded into Stata, the **bbandits_initialize** command can be used to generate the required data structure and separate the list of subjects into equally sized batches. Additionally the number of treatment arms and the exploration phase has to be specified. The exploration phase defines the number of batches where the treatment is just uniformly distributed.

The **bbandits_initialize** function will randomly assign the treatment arms in the specified exploration phase. For our simulated example, consider 1,000 school teachers, who will be separated into ten batches with an exploration phase of two batches. In Figure 7 the generated data structure is shown. Each ID corresponds to a level of the variable *chosen_arm*. Since we have not collected any data for the reward yet, the variable *reward* is set to missing. Once observations come in, this variable holds the realized rewards from the assigned treatments. For example, treatment 3 (surveillance camera) is assigned to *school_1*. The observed reward (attendance of the teacher) needs to be recorded in the column for the variable *reward* for *school_1*. Application of the Sequential Arm Elimination algorithm described in section 2.1 requires inclusion of the `SAE` option.

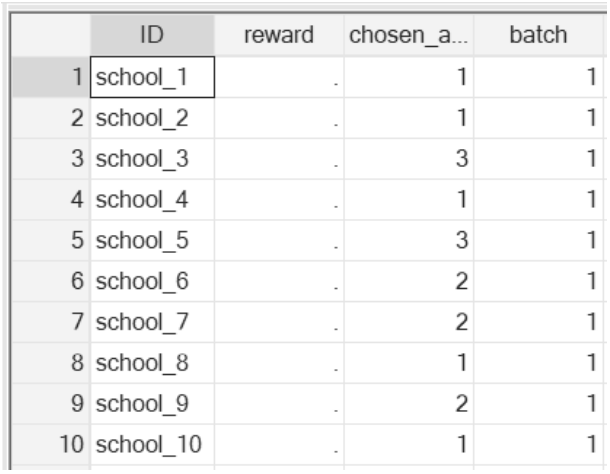
```
. bbandits_initialize, batch(10) arms(3) exploration_phase(2)
```

7.2 After initialization

In the next step the user has to run the first batches of the experiment and observe the rewards (outcome). To produce a new assignment scheme based on the realized rewards so far, the data including the newly observed rewards need to be reloaded into Stata.

The **bbandits_update** function can now be used to assign the treatment according to a bandit algorithm for the next batch. In our example, the outcome variable is binary, so we could specify the Bernoulli Thompson Sampling algorithm with the option *Thompson*. Additionally, the *clipping* parameter has to be specified for the Thompson Sampling algorithm. For the ε-greedy algorithm specifying the parameter ε is required. To apply the Sequential Arm Elimination (SAE) algorithm (see section 2.1), the number of active arms, the batch structure, and the `SAE` option must be specified.

Figure 7: Initial data structure



	ID	reward	chosen_a...	batch
1	school_1	.	1	1
2	school_2	.	1	1
3	school_3	.	3	1
4	school_4	.	1	1
5	school_5	.	3	1
6	school_6	.	2	1
7	school_7	.	2	1
8	school_8	.	1	1
9	school_9	.	2	1
10	school_10	.	1	1

```
   . bbandits_update reward chosen_arm batch, thompson clipping(0.2)
> excel("mypath")
```

The *clipping rate* sets the minimum probability that an arm is played. Without setting a clipping rate, inference can become a problem because in some batches, some treatment arms may not be assigned which leads to a non-defined margin in these batches. Regardless of the chosen algorithm, the **bbandits_update** function will return i) the treatment arms which should be played in the next batch and ii) the according treatment probabilities for each arm. Now, the user has to assign the treatment to the treatment arms according to the algorithm and observe the results. Then, the observed rewards are read into Stata. The easiest way to implement this procedure is to save the data set into a file and then add the rewards to the file. This can either be done in Stata or in a different program. The **bbandits_update** function has the option *Excel* which saves the data set into an Excel file in the specified folder. For small-scale experiments one convenient option would be to record the rewards in an Excel file and then read the file back into Stata. Figure 8 shows the data structure exported in Excel where the rewards can be recorded in the shaded cells. The *chosen_arm* column in the figure displays which treatment arm is assigned according to the bandit algorithm. The *chosen_arm_numeric* column will automatically be generated and entails an integer which corresponds to the "chosen_arm" category to facilitate the transfer to other programs. The user has to fill-in the observed rewards into the variable *rewards* and then load the file back into Stata.

Subsequently, the **bbandits_update** is applied again and all steps are repeated until the experiment is completed. Finally, the results can be analyzed with the **bbandits** command.

Figure 8: Data structure for updating

	A	B	C	D	E	F
1	ID	rand	reward	chosen_arm	batch	chosen_arm_numeric
192	school_19	1	0	1	2	2
193	school_19	1	1	3	2	0
194	school_19	0	1	3	2	0
195	school_19	1	0	3	2	0
196	school_19	0	1	2	2	1
197	school_19	0	1	1	2	2
198	school_19	1	0	2	2	1
199	school_19	1	0	3	2	0
200	school_19	0	1	2	2	1
201	school_20	0	1	1	2	2
202	school_20	1		3	3	0
203	school_20	1		3	3	0
204	school_20	1		3	3	0
205	school_20	1		2	3	1
206	school_20	0		2	3	1

8 Conclusions

This article demonstrates how batched bandit designs can make adaptive experiments both practically feasible and statistically credible. Beyond the specific estimators and algorithms discussed, our results highlight several broader implications for future applied work.

First, adaptive experiments can substantially reduce the ethical and financial costs of experimentation, but only when implemented with careful attention to inference. Our findings indicate that the number of observations per batch is the key determinant of unbiased estimation and valid inference. This insight suggests that future experimental designs should prioritize sufficiently large batch sizes, especially in settings with small effect differences.

Second, the comparison of OLS and BOLS estimators underscores the importance of methods that remain robust under adaptive sampling. While both estimators are consistent, their performance differs across effect sizes and assignment dynamics. Researchers designing new adaptive experiments may therefore benefit from simulation-based design analysis before data collection, using tools such as `bbandits` to assess estimator behavior under plausible scenarios.

Third, the `bbandits` command lowers the barrier to implementing adaptive experiments in practice. A promising direction for future research is to extend these tools to settings with covariate-adaptive assignment, contextual bandits, or non-stationarity, enabling richer experimental designs.

9 Acknowledgments

We thank Guillaume Bied, Johannes Gaul, Morgane Hoffmann, Michael Knaus, Charly Marie, Bertille Picard, David Preinerstorfer, and Thomas Simon, and participants at the 2025 Oceania Stata Conference for valuable comments and discussions. We declare that we have no interests, financial or otherwise, that relate to the research described in this paper. We are grateful to the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) for financial support through CRC *TRR 266 Accounting for Transparency* (Project-ID 403041268).

10 References

Agarwal, D., B. Long, J. Traupman, D. Xin, and L. Zhang. 2014. LASER: A Scalable Response Prediction Platform for Online Advertising. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, 173–182. WSDM '14. New York, NY, USA: Association for Computing Machinery.

Ahrens, A., C. B. Hansen, M. E. Schaffer, and T. Wiemann. 2022. Stata ML Page. Online documentation, available at https://statalasso.github.io/docs/python.

Athey, S., and G. W. Imbens. 2019. Machine Learning Methods That Economists Should Know About. *Annual Review of Economics* 11: 685–725.

Avivi, H., P. Kline, E. Rose, and C. Walters. 2021. Adaptive Correspondence Experiments. *AEA Papers and Proceedings* 111: 43–48.

Bibaut, A., and N. Kallus. 2025. Demystifying Inference After Adaptive Experiments. *Annual Review of Statistics and Its Application* 12: 407–423.

Bischof, J., P. Doerrenberg, D. Rostam-Afschar, D. Simons, and J. Voget. 2025. The German Business Panel: Firm-Level Data for Accounting and Taxation Research. *European Accounting Review* 34(4): 1499–1527.

Burtini, G., J. Loeppky, and R. Lawrence. 2015. A Survey of Online Experiment Design with the Stochastic Multi-Armed Bandit. arXiv 1510.00757, [stat.ML].

Camerer, C., D. Bose, R. Daviet, and T. Imai. 2024. Social Preference Estimation Using Adaptive Experimental Design. Working paper. California Institute of Technology.

Caria, A. S., G. Gordon, M. Kasy, S. Quinn, S. O. Shami, and A. Teytelboym. 2023. An Adaptive Targeted Field Experiment: Job Search Assistance for Refugees in Jordan. *Journal of the European Economic Association* 22(2): 781–836. https://doi.org/10.1093/jeea/jvad067.

Chapelle, O., and L. Li. 2011. An Empirical Evaluation of Thompson Sampling. In *Advances in Neural Information Processing Systems*, ed. J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger. Vol. 24. Curran Associates, Inc.

Chapman, J., E. Snowberg, S. W. Wang, and C. Camerer. 2024. Dynamically Optimized Sequential Experimentation (DOSE) for Estimating Economic Preference Parameters. Working Paper 33013, National Bureau of Economic Research.

Chen, J., and I. Andrews. 2023. Optimal Conditional Inference in Adaptive Experiments. arXiv 2309.12162, [stat.ME].

Duflo, E., R. Hanna, and S. P. Ryan. 2012. Incentives Work: Getting Teachers to Come to School. *American Economic Review* 102(4): 1241–1278.

Esfandiari, H., A. Karbasi, A. Mehrabian, and V. Mirrokni. 2021. Regret Bounds for Batched Bandits. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI-21)*, 7340–7348. Association for the Advancement of Artificial Intelligence.

Gaul, J. J., F. Keusch, D. Rostam-Afschar, and T. Simon. 2025. Invitation Messages for Business Surveys. A Multi-Armed Bandit Experiment. *Survey Research Methods* 19(4): 409–429.

Graepel, T., J. Q. Candela, T. Borchert, and R. Herbrich. 2010. Web-Scale Bayesian Click-Through Rate Prediction for Sponsored Search Advertising in Microsoft's Bing Search Engine. In *Proceedings of the 27th International Conference on International Conference on Machine Learning (ICML '10)*, 13–20. Omnipress, Madison, WI, USA.

Hack, L., and D. Rostam-Afschar. 2024. Understanding Firm Dynamics with Daily Data. CRC 224 Economic Perspectives on Societal Challenges Discussion Paper 593 and CRC 266 Accounting for Transparency Working Paper Series No. 155.

Hadad, V., D. A. Hirshberg, R. Zhan, S. Wager, and S. Athey. 2021. Confidence Intervals for Policy Evaluation in Adaptive Experiments. *Proceedings of the National Academy of Sciences* 118(15): e2014602118.

Hill, D. N., H. Nassif, Y. Liu, A. Iyer, and S. Vishwanathan. 2017. An Efficient Bandit Algorithm for Realtime Multivariate Optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '17. ACM.

Hirano, K., and J. R. Porter. 2025. Asymptotic Representations for Sequential Decisions, Adaptive Experiments, and Batched Bandits. arXiv 2302.03117, [econ.EM]. https://arxiv.org/abs/2302.03117.

Hoffmann, M., B. Picard, C. Marie, and G. Bied. 2023. An Adaptive Experiment to Boost Online Skill Signaling and Visibility. Working paper, CREST - ENSAI. Accessed: 2024-04-10. https://bertillepicard.github.io.

Kasy, M., and A. Sautmann. 2021. Adaptive Treatment Assignment in Experiments for Policy Choice. *Econometrica* 89(1): 113–132.

Kemper, J., and D. Rostam-Afschar. 2025. Inference for Batched Adaptive Experiments. Working Paper Series 214, CRC 266 Accounting for Transparency.

Lei, H., Y. Lu, A. Tewari, and S. A. Murphy. 2022. An Actor-Critic Contextual Bandit Algorithm for Personalized Mobile Health Interventions. arXiv 1706.09090, [stat.ML].

Offer-Westort, M., A. Coppock, and D. P. Green. 2021. Adaptive Experimental Design: Prospects and Applications in Political Science. *American Journal of Political Science* 65(4): 826–844.

Perchet, V., P. Rigollet, S. Chassang, and E. Snowberg. 2016. Batched Bandit Problems. *Annals of Statistics* 44(2): 660–681.

Rafferty, A., H. Ying, and J. Williams. 2019. Statistical Consequences of using Multi-armed Bandits to Conduct Adaptive Educational Experiments. *Journal of Educational Data Mining* 11(1): 47–79.

Schulz, E., N. T. Franklin, and S. J. Gershman. 2020. Finding Structure in Multi-Armed Bandits. *Cognitive Psychology* 119: 101261.

Scott, S. L. 2010. A Modern Bayesian Look at the Multi-Armed Bandit. *Applied Stochastic Models in Business and Industry* 26(6): 639–658.

———. 2015. Multi-Armed Bandit Experiments in the Online Service Economy. *Applied Stochastic Models in Business and Industry* 31: 37–49. Special issue on actual impact and future perspectives on stochastic modelling in business and industry.

Sutton, R. S., and A. G. Barto. 2018. Reinforcement learning: An introduction. *A Bradford Book* Available at http://incompleteideas.net/book/the-book-2nd.html.

Thompson, W. R. 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika* 25(3-4): 285–294.

———. 1935. On the Theory of Apportionment. *American Journal of Mathematics* 57(2): 450–456.

Zhan, R., V. Hadad, D. A. Hirshberg, and S. Athey. 2021. Off-Policy Evaluation via Adaptive Weighting with Data from Contextual Bandits. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2125–2135. KDD '21. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3447548.3467456.

Zhang, K., L. Janson, and S. Murphy. 2020. Inference for Batched Bandits. *Advances in Neural Information Processing Systems* 33: 9818–9829.

About the authors

Jan Kemper is a PhD student at the University of Mannheim and at the ZEW.

Davud Rostam-Afschar is Professor at the University of Mannheim, affiliated with IZA, GLO, and NeSt.