Pitfalls and Tradeoffs in Implementing the Right to be Forgotten

ABSTRACT

Right to be Forgotten (RTBF) in one of the oldest and prominent of the data rights. While its legal intention is straight forward (for example, the GDPR describes it in just 417 words), the computing community has found it challenging to implement this in practice. For example, regulators have issued 216 RTBF violations in the first five years of GDPR i.e., an RTBF failure once every 8 days, on average. In this work, we identify the uncertainties and risks in supporting RTBF from a computing perspective, and then propose a layered approach towards mitigating those. To ground our research, we design and implement RTBF capability into Elasticsearch, a popular open-source search engine. Our work is the first to establish a rich tradeoff space in RTBF compliance as well as recognize the presence of RTBF anti-patterns in the real-world.

1 INTRODUCTION

> *"For every complex problem, there is an answer that is clear, simple, and wrong."*
>
> Henry Louis Mencken

Ancient wisdom says that everything that has a beginning has an ending. However, when it comes to the lifecycle of personal data, the ending was nowhere in sight. In fact, for much of its existence, the computing community has evolved without treating deletion as a first-class operation. This practice had to change when the European Union introduced the Right to be Forgotten (RTBF)—first in 2014, as a stand alone right applicable only to online search engines and then in 2018, as a universal right applicable to all data controllers through the General Data Protection Regulation (GDPR) [2].

"Isn't it just deletion?" has been the computing community's standard reaction to the requirements of RTBF. While the end goal of RTBF is indeed the deletion of data, casting RTBF as *just deletion* is akin to saying that eating is only for nutrition or sex is all about reproduction. It is not surprising that over the last five years, an RTBF penalty is issued once every eight days—a clear sign that the computing and data management communities have continued to oversimplify, misunderstand, and ill-implement RTBF. Our work is an attempt to remedy this disconnect.

We demonstrate how RTBF exposes computing systems to uncertainties and challenges at all stages of design and operation (in Section-2.2), and how RTBF has invalidated principles and practices of data management with decades of precedent (in Section-3.3). To mitigate these impact, we propose a principled approach for designing RTBF capability in computing systems (in Section-3). We ground our work firmly in both law and computing—the former, by basing our analysis on GDPR and the first five years of its enforcement, and the latter, by implementing our findings in Elasticsearch, a widely deployed search engine. This approach has helped us identify novel insights and make evidence-based recommendations that previous domain-specific investigations have lacked. In particular, we make three key contributions:

- **Designing End-to-End RTBF Capability.** We propose a two-phase approach for introducing RTBF-capability into applications. First, in *law-driven design*, we analyze the legal requirements of RTBF and map them into specific computing and data management tasks. Then, in *enforcement-driven refinement*, we analyze the enforcement decisions to identify the tradeoffs and choices in implementing those tasks.
- **RTBF Anti-Patterns.** RTBF anti-patterns are the long-standing principles and practices of computing and data management systems that serve their original purpose well, but make it difficult to support RTBF. After analyzing the first five years of RTBF enforcement in Europe, we identify six such RTBF anti-patterns (summarized in Table 1 and discussed in Section-3.3).
- **RTBF-Capable Elasticsearch.** We analyze the RTBF capabilities of Elasticsearch and identify the functionalities that are lacking. Then, we demonstrate the tradeoff between different profiles of RTBF compliance (optimized for cost, performance, or risk) and benchmark Elasticsearch using Rally. We publicly release all our software artifacts and datasets at https://github.com/lawfulcomputing/RTBF.

2 BACKGROUND AND MOTIVATION

In this section, we discuss the importance of the problem, review related work, and establish the need for and novelty of our work.

2.1 RTBF

Right to be Forgotten, alternatively referred to as Right to Erasure, is the right of an individual person to request an organization to delete their personal data. RTBF gives people an ability to prevent others from seeing their personal information that they deem old, inappropriate, irrelevant, or simply prefer to make private. This right is distinct from right to privacy [1], which prevents governments and other entities from forcing a person to reveal their personal data; instead, RTBF is to be used for information that is already public, but the individual no longer wants it to remain public.

RTBF became a legal right for the first time in 2014. In a case against Google [17], the European Court of Justice ruled that European people can request search engines to remove certain links from their search results, if their privacy concerns outweigh the public interest in the information contained in those links. Later, when GDPR went into effect in 2018, RTBF was expanded to cover all data controllers, not just search engines. However, GDPR did not make RTBF an absolute right i.e., for an RTBF request to be honored, it has to meet one of the six conditions and not fall under one of the five exemptions (we show these in Figure 1, where we produce GDPR article 17 verbatim). To keep our discussion concrete, we focus exclusively on GDPR's version of RTBF and prefix all articles of GDPR with $\mathscr{G}$.

Table 1: RTBF Anti-Patterns in computing and data management systems

	RTBF anti-patterns	Real-world examples
1	Using personal data as primary keys	*Copenhagen company unable to support fine-grained RTBF due to database constraints [10]*
2	Anonymizing data instead of deleting	*AXA anonymizing personal data to preserve its knowledge beyond expiry date [28]*
3	Keeping personal data untagged	*Clearview AI requiring users to provide an exact copy of the photo to be deleted [29, 54]*
4	Mismanaging the depth of deletion	*Google Search propagating RTBF requests to websites that published the original URL [11]*
5	Logging without checks and bounds	*Belgian company storing excessive information about RTBF requests in their logs*
6	Employing excessive verification	*Twitter requiring photo ID for RTBF while allowing account creation without it [25]*

The need for RTBF arose in early 21st century when organizations began collecting personal information at scale, and search engines made these accessible globally. As Viktor Mayer-Schonberger has chronicled in his book [49], throughout the human history, forgetting was the norm and remembering was the exception. Given how this phenomenon gotten largely reversed in the recent decades, RTBF is hailed a countermeasure against this trend. That said, RTBF has received criticisms [26, 30, 36, 67, 69] as a means to rewrite history, weaken the freedom of expression, enable censorship and other less desirable social outcomes. While this is an important debate for our society, the focus of our work is in exploring the challenges that computing systems face in implementing RTBF and how to systematically solve them.

2.2 Challenges in Complying with RTBF

When new regulations are enacted, legal experts and policy makers tend to limit their expositions to core legal principles that are broadly interpretable and will hold the test of time. While legally prudent, this strategy makes it challenging for computing systems to adapt and support regulations such as RTBF. These challenges can come in the form of lack of precise specifications, undefined tradeoffs in performance-vs-risk, uncertainties in managing new technologies, among others. We illustrate how such challenges can manifest at different stages of system design and operation:

Examples at architecture level. Systems that are architected with little or no prior consideration for RTBF, find it hard to add that capability later. For example, an organization that uses a personal data as a primary key in their databases, would find it tricky to delete that item. Problems could also stem from unwise choices in organizing the data. For instance, Clearview AI built their facial recognition system by training on billions of images from the Internet. However, when people approached them with RTBF requests, they realized that they could not identify all the photos that belong to a given person (since they had not tagged the images at the time of collecting or processing). They were fined in 2022 by multiple regulators [20, 29, 54] for this limitation. More generally, RTBF in machine learning systems is a nascent area of research with no generic or efficient solutions that can help models forget a select data in their training set.

Examples at design and implementation level. RTBF opens up many uncertainties and unknown tradeoffs at the systems level. Consider the *latency of deletion* i.e., how soon after the request, should the data be removed. Designers could opt for a strict compliance by making deletions synchronous in real-time, or choose a relaxed compliance by allowing deletions to happen eventually. Prior work [63] has shown the effect of synchronous deletion on two popular database systems, Redis and PostgreSQL, both of which experienced a slowdown of up to 20%. On the other hand, eventual compliance allows stale data to linger in the system for unspecified amount of time, posing security and privacy risks. Next, consider the *depth of deletion* i.e., should the data be deleted from all memory and storage subsystems going all the way to the hardware, or simply be forgotten at the service level. While the former leads to a strict form of compliance, it adds significantly to the latency and complexity of the deletion process. The latter, however, exposes the organization to legal risks since other services may unwittingly end up using the said data. GDPR does not offer clarity on many such systems level issues.

Examples at operation level. RTBF is not an absolute right i.e., just because an RTBF request is made on valid personal data by a verified data subject, does not mean that it should be honored. GDPR requires all controllers to balance the rights of individuals with the interests and obligations to the society. This is challenging for organizations since it turns RTBF from a *generic operation* to a *highly individualized process*, thereby making it hard to fully automate it. The gravity of this challenge is evident when you consider that RTBF is operated largely as a manual process at Google and Microsoft [9, 50] (organizations that are considered technologically sophisticated), and that they take about 6 days, on average, to arrive at an RTBF decision. These challenges are so pervasive that 41% of all RTBF violations are due to incorrectly interpreting the validity of RTBF requests [64].

2.3 Related Work

RTBF in computing systems. Several organizations including Google [9], Microsoft [50], and Wikipedia [34] have shared details about how their applications support RTBF. These reports primarily focus on aggregate-level characterization of the received RTBF

requests and their responses, but do not offer much details on the technical or policy aspects of their internal RTBF implementation.

Orthogonal to this perspective, researchers have explored how people perceive RTBF support on social media websites [40, 51], and the challenges they face while exercising RTBF [39, 65]. This body of work focuses on human-computer interfacing for RTBF, and treats the target computing systems and services as black boxes. In contrast, we focus on designing and implementing computing systems that can support RTBF.

Deletion in computing systems. Deletion is one of the fundamental operations of database systems (as represented in *CRUD*, the venerable acronym for *Create-Read-Update-Delete*), yet it had long been treated as a second-class operation. However, GDPR and the onset of data rights has brought the attention of the computing community back to deletion. Key advances in the last five years include: Google cloud publishing their deletion pipeline and guaranteeing to erase all copies of the data within 180 days of requesting [3]; Facebook designing a system that can assure correctness of deletion [22]; researchers from Boston University building a key-value database that lets users control deletion latency [61]; and Berkeley's cryptographic framework for data deletion [35].

Given the important role training data plays in machine learning, the notion of machine unlearning i.e., making ML systems forget all they learnt from a given data, has gained traction in the AI/ML community [16]. These work primarily focus on making the deletion efficient for certain models [13, 46, 68] or specific applications [18, 37]. Lastly, secure data deletion i.e., deleting data irrevocably from a physical storage, has been an active area of research in file systems, operating systems, and backup systems [8, 31, 41, 56–58].

A key trait of all these work is that they use RTBF as a motivation to implement deletion in a target computing system (say, databases, file systems, cloud computing, machine learning systems, etc.,), but they largely abstract out the legal and policy aspects of RTBF. A central thesis of our paper is that RTBF is not just deletion, and by ignoring the interplay between law and computing, these prior work fail to recognize important system design issues and tradeoffs.

3 DESIGNING RTBF

We propose a novel two-phase approach to designing end-to-end RTBF capability. This is motivated by a core dichotomy between computing and law—namely, while computing applications are created to be precise and specific, laws are written to be abstract and interpretable—and our attempt to bridge this divide. In phase-1, we analyze the *language of the law* (which changes rarely) and map it to a set of high-level computing tasks. In phase-2, we examine the *enforcement of the law* (which evolves frequently) to identify available design choices and tradeoffs, and to weed out non-compliant ones.

3.1 Phase-1: Law-driven Design

Mapping Legal Intentions to Computing Tasks. By analyzing RTBF from a computing perspective, we identify four key tasks that cover all of its legal requirements. Figure 1 highlights this process by showing the legal text, the computing tasks, and the mapping between the two. At this stage, it is important to keep the RTBF tasks high-level so that we do not lose the generality and interpretability afforded by the law, while at the same time, we set up a foundation to methodically explore low-level design and implementation choices in phase-2.

- **Interface with data subjects.** This task encompasses all the input output operations of the RTBF machinery. An RTBF capable system must provide a mechanism for people to submit an RTBF request and to get a response whether the said data was forgotten (and if not, an explanation for not doing so). While $\mathscr{G}17$ does not require any particular modalities for interfaces, other articles stipulate that interfaces should not impose additional burden for exercising data rights. Commonly offered interfaces for RTBF include web forms, mobile apps, email, postal mail, and telephone. In practice, the interfacing task also filters out spurious RTBF requests by authenticating the person making the request.
- **Establish deletion policies.** RTBF is not an absolute right. For an RTBF request to be honored, it has to meet at least one of the six conditions (laid out in $\mathscr{G}17.1$a through 1f), and not fall under any of the five broad exemptions (specified in $\mathscr{G}17.3$a through 3e). The goal of this task is to resolve this contention and produce a yes or no decision (i.e., whether to honor an RTBF request or not). In real world, this task has proved quite challenging to be fully automated. For instance, Google Search still manages its policy resolution as a human-driven process, taking 6 days, on average, to resolve a request [9]. This task should also govern if data that is shared with external applications and datastores should also be deleted.
- **Erase data from data processing systems.** This task is responsible for erasing data from all software and hardware systems that process data including application software, libraries and system software, operating systems, hardware processors, cloud and other external processing systems. In order to make data processing performant and reliable, these systems may keep data in their in-memory data structures, in caches and runtime engines, in logs, in networked and remote processes, among others. So, the goal of this tasks is to propagate the deletion request programmatically to all subsystems (both internally and externally) and to ensure that data is deleted in a timely manner. The main challenge from a computing perspective is that many subsystems lack native support for fine-grained data deletion, and many systems implement them as lazy or shallow deletes.
- **Erase data from data storage systems.**[1] The final task is to erase data from software and hardware systems that offer long-term, persistent storage for data. These include database systems, file systems, cloud storage, backup systems, among others. The key challenge from computing perspective is that most systems are optimized for performance, scalability, and reliability, which makes data deletion a second-class operation.

[1] GDPR does not demarcate between systems that store data vs. those that process data. However, from a computing perspective, this distinction is significant. For e.g., they are different subfields of the domain. So, we explore them under two distinct tasks.

Figure 1: Mapping RTBF's legal requirements into computing domain. We identify four key tasks of RTBF capable systems (on the right), and the text of the regulation that requires each of these (on the left).

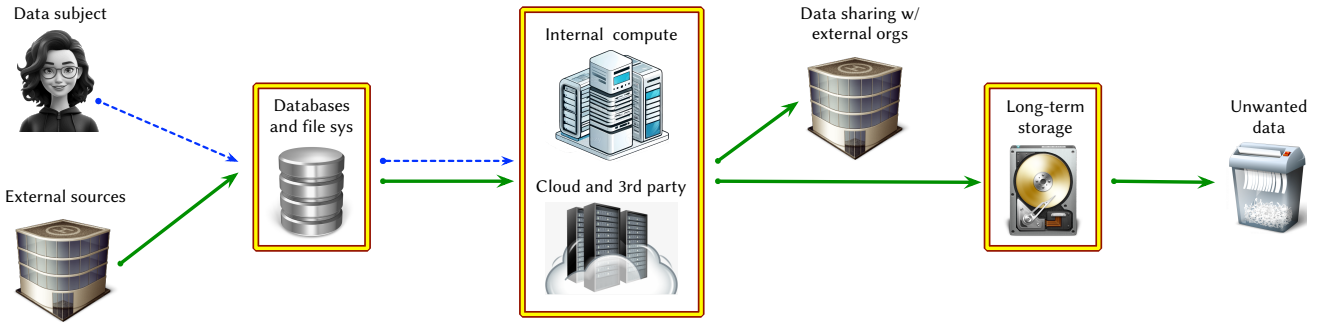


Figure 2: Demonstrating the sufficiency of the RTBF tasks. Figure shows the full lifecycle of data from (left to right) acquiring, storing, processing, sharing, archiving, and to destroying. Then, we show how the four RTBF tasks (represented by blue dotted arrow, green solid arrow, and red-yellow double lines) can propagate and implement deletion at any of the lifecycle phases).

This manifests in not having APIs to delete data from certain subsystems, not offering time guarantees on deletions, etc.

Necessity and Sufficiency of RTBF Tasks. We want to establish that the four identified RTBF tasks are both necessary and sufficient to meet the legal obligations of RTBF. To ascertain the necessity of the four RTBF tasks, we employ proof by contradiction. Consider an RTBF-compliant system that does not perform *interfacing with data subjects*. In such a system, no data subject can send their request to be forgotten, nor would they hear back from the system

if their data was erased. This would be a violation of the legal requirements of RTBF (as stated in the first sentence of $\mathscr{G}17$), in turn contradicting our assumption that the system was RTBF compliant. Next, let us consider an RTBF-compliant system that does not have *established policies for deletion*. In absence of policies, such a system has to blindly honor all RTBF requests or summarily reject all RTBF requests. However, the first approach violates the legal requirement that any exercising of the individual rights must be balanced against the public interest (as stated in $\mathscr{G}17.3$(c)) and the latter approach simply devoids data subjects of their RTBF rights. Thus, absence of policies would make the system non-compliant with RTBF, a contradiction. Finally, consider an RTBF-compliant system that does not have the ability to *erase data from application software* or the capability to *erase data from data management systems*. When such a system receives a legitimate RTBF request that must be honored, it will be unable to meet its obligation of erasing data without undue delay (as specified in $\mathscr{G}17.1$), thereby invalidating the original assumption on RTBF compliance. Thus, we have established the necessity of the four RTBF tasks in achieving RTBF capability.

To establish sufficiency, we consider the lifecycle of data from a computing perspective. As shown in Figure 2, starting from the left, data is acquired either directly from data subject or from another controller; then it is stored in databases, file systems, and other short- to medium-term storage systems; third, it is consumed by data processing infrastructure; optionally, shared with external organizations; then it may be archived for long-term storage and retrieval; and finally, when it is no longer needed, it gets destroyed. Since all data have to be in one or more of these phases (by definition of the lifecycle), if we can show that the four RTBF tasks can propagate deletion request and bring about deletion in all these phases, then we would have established the sufficiency condition. First, we see that the RTBF tasks three and four (i.e., *erase data from data processing systems* and *erase data from data storage systems*) naturally cover the store, process, and archive phases (as marked in double yellow-red lines). Next, the RTBF task two i.e., organization's deletion policy, can specify rules for how an RTBF request is propagated from one phase to another, how each component responds to such a request, and how to handle any failures. We indicate this control flow in solid green arrows in Figure 2. Finally, RTBF task one, namely *interface with data subject*, can propagate deletion requests from data subjects to organization's storage and compute phases, and convey their responses back to data subjects (as seen by dotted blue lines). Thus, we have established that the four RTBF tasks are sufficient to cover all the deletion-related control and data flows in the full lifecycle of data.

3.2 Phase-2: Enforcement-driven Refinement

Tracking the Enforcement of GDPR. GDPR follows a distributed model of enforcement. Though the law is legislated centrally by the European parliament, its implementation is left to member nations, each of whom must enforce it within their countries. Thus, on the ground, GDPR is enforced by ~28 independent and distributed agencies called the Data Protection Authorities or DPAs. Each DPA operates autonomously, determining their own priorities and enforcement strategies, and working within the budget allotted by its national government. Finally, to avoid significant divergence in the application and interpretation of GDPR across countries, the law has set up a trans-national agency called the European Data Protection Board (EDPB), which can offer biding rules to DPAs on disputed issues.

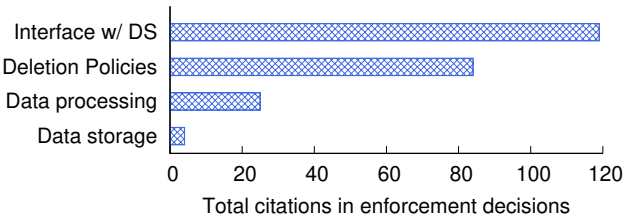


Figure 3: Distribution of citations across four RTBF tasks

There are several projects that track the enforcement of GDPR by crawling, collecting, and annotating enforcement decisions by DPAs and EDPB. Prominent ones include GDPRhub [62], Enforcement Tracker [45], and GDPRxiv [64]. We decided to use the corpus from GDPRxiv since it is based on an open methodology and also has the largest number of enforcement decisions across all repositories. While there are 4206 enforcement decisions in the first five years of GDPR (between 25-may-2018 to 25-may-2023), we are interested in 216 of them that cite $\mathscr{G}17$. Our analysis and findings in the rest of the section are based on these RTBF related decisions.

Understanding RTBF Enforcements in the Field. RTBF enforcements have been issued, on average, once every 8.4 days. About 55% of these enforcements are warnings and criticisms that do not impose a financial penalty (these tend to be cases where the violation is a first-time offense, did not affect a large number of the people, and where the controller cooperated with the DPA to fix their violation). Amongst the cases that were levied a financial penalty, the highest was €27.8M by the Italian DPA on the telecommunication company TIM Group and the average penalty over the five year period was €1.09M. Google received the most enforcements of any organization at a count of six. While these are interesting statistics about RTBF enforcement, our main goals are: (i) to analyze the failures from a computing perspective, and (ii) to identify how these enforcements influence the design choices and tradeoffs available for computing systems.

First, for each RTBF decision, we identify the main reason(s) for failure, and then map it to one or more of the four foundational blocks of RTBF (as discussed in Section-3.1). Since this annotation is too big to be included in the paper, even in the appendix, we make it available on the project website. Figure 3 plots this distribution with the number of citations on the X-axis and the corresponding RTBF task they failed on the Y-axis. We see that 55% of enforcements cite failures in interfacing with data subjects, while 38% cite failures in policy resolution. While this distribution is skewed, it is not entirely surprising since UI and policy failures are easier to catch and they impact people immediately (as opposed to failures in data processing and data storage systems). Next, we analyze enforcement decisions, both individually and collectively, to identify the design and implementation aspects of RTBF that are influenced by them. We present five such findings:

- **Response Time.** It measures the time between making an RTBF request and receiving the first response from the controller. The recommended value comes from $\mathscr{G}12(3)$, which states that all data right requests should be responded within one month of receipt (while allowing exemptions for cases requiring complex processing). Yet, this is one of the most cited reasons for RTBF failures, accounting for ~35% of all enforcements. Enforcements have clarified the minimum expectation: controllers must acknowledge RTBF requests within a month of receipt and explain how they would be handled; though RTBF decision-making and actual data deletion may take longer.
- **Explainability.** This measures the ability of a data subject to understand how their RTBF request was handled. While the law is not explicit on how in-depth or individualized the explanations have to be, the enforcements have shed light on what is insufficient. First off, it is clear that explanations are important when rejecting RTBF requests. For instance, in two separate cases against Google and Spotify, regulators agreed that the companies were correct in rejecting the RTBF requests but fined them for lacking proper explanation in their responses. Secondly, precedents indicate that the controllers must explain the key reasoning behind their decision. This could be due to influence of other laws (for instance, RTBF cannot be exercised on betting data since online betting laws require this data to be kept for five years—Bet365 in Denmark); legal obligations (say, the controller keeps a log of all RTBF requests and no RTBF will be supported on that log itself—unnamed company in Belgium); or business practices (such as not allowing RTBF on payment default data unless the debt has been paid off—unnamed company in Hungary). Straight forward as these are, none of these explanations were included in the RTBF responses, and all three controllers were fined or warned for omitting them.
- **Deletion latency.** Deletion latency is the time between a controller approving an RTBF request and actually deleting the data. As such, this metric is unrelated to and unaffected by response time. While the law simply specifies "without undue delay" in $\mathscr{G}17(1)$, the enforcements offer more clarity. At one extreme, French regulators fined the retailer Brico Prive for simply deactivating user accounts upon receiving RTBF requests instead of actually deleting their data—establishing that deletion latency cannot be infinity. While on the other hand, Austrian regulators accepted the practice of purging the deleted data in batches once every year as long as the controller commits to stop using that data immediately. Given these precedents, it would be prudent to have a well-defined schedule for purging the data, taking into account the underlying technical systems, and then sharing this number with the users. An example of this is Google cloud's committment to thoroughly delete all RTBF data within 180 days [3].
- **Exemption handling.** Accommodating exemptions in RTBF decision-making has proven to be a difficult task in the field (accounting for 38% of all enforcements). A key question from computing perspective is if this could be fully automated. The volume of failures and number of exemptions cited in them indicate that we may be far from that. Thus far, 41 cases say exemptions from within GDPR were misinterpreted, 24 cases say the interplay of RTBF with non-GDPR laws was disregarded, and 13 cases say both were incorrectly handled. So, it would seem that Google search's approach of using skilled human arbiters is not just a risk-averse approach but indeed the state of the art in policy resolution [9]. Also, it must be noted that, so far, we have not seen a case where a controller was penalized for honoring an RTBF request that should have been rejected. So, it may be prudent to weigh more in favor of honoring the RTBF request unless the applicability of exemptions is glaringly obvious.
- **Deletion granularity.** This metric represents the standard unit of data on which RTBF could be exercised. At one extreme, a controller could restrict RTBF to individual items of data. This is appealing for controllers since it makes the identification and deletion of data straight forward. However, since it puts extra burden on people exercising RTBF, this practice was deemed insufficient in a case against Clearview AI, which required people to submit the exact copy of the photos to be deleted (as opposed to just the person's identity) [54]. The other extreme of granularity is the all or nothing approach, where the controller only supports deleting all data belonging to a person. In two separate cases, regulators in Austria and Denmark, have stated that technical constraints arising from database management systems cannot be used to restrict users from deleting select items of data. In light of these enforcements, a safe middle ground would be multi-granular RTBF. For example, Google Search allows deleting search history by individual queries, by a date range, or the entire search history.

Scope and limitations. Unlike phase-1 of the design, phase-2 is not comprehensive i.e., several aspects of design and implementation may still be open to broad interpretations unless and until regulators and courts rule on them. Also, phase-2 is never complete: as new enforcements are issued, more clarity emerges on the design aspects and new thresholds are established. This makes RTBF compliance a moving target, and requires systems to continuously albeit periodically evaluate their design choices and operating practices to ensure consistency with new precedents. Finally, in presenting these five findings, we focus on aspects that are broadly applicable and not specific to particular domains like healthcare. We also exclude findings that have little or no technical bearing, say controllers refusing to accept RTBF requests.

3.3 RTBF Anti-Patterns

Anti-patterns are principles and practices of software design that seem intuitive and useful in a narrow context, but prove ineffective in a broader system and over long term. Examples of anti-patterns include premature optimization of functions, use of magic numbers, and creating a god class in object design, among others. The idea of anti-patterns was originally conceived by Andrew Koenig [43] and later expanded by Brown et. al., [15]. In a similar vein, we define *RTBF anti-patterns* as the long-standing principles and practices of computing systems that serve their original purpose well, but make

it harder to support RTBF. After analyzing the RTBF enforcement corpora, we have identified five such RTBF anti-patterns:

1. Using personal data as primary keys

Personal data, especially those that uniquely identify people, say phone numbers, email, or national IDs, have long been used as primary keys in database systems. While GDPR does not preclude this practice, it often results in practical difficulties in honoring RTBF. For instance, when the Danish regulators found that Taxa (a Copenhagen based taxi company) had difficulties in removing user data because of database systems constraint, they were fined and asked to redesign their data storage systems. Similarly, Carrefour (a French retail company) resorted to using ad hoc tools to circumvent database system constraints during an investigation by the French regulator, ultimately resulting in €2.25M penalty.

2. Anonymizing personal data instead of deleting

Organizations may chose to anonymize the personal data upon getting an RTBF request [7, 23, 55]. This is appealing because it lets the controller derive some benefit from the data that otherwise would be permanently deleted. However, a robust anonymization is hard to achieve. For example, in 2019, the Danish DPA fined a taxi company for anonymizing its 9 million taxi ride records while retaining enough auxiliary data to be able to reconstruct the dataset [10]. Even when it is well done, prior research has shown that not all anonymizations hold the test of time [53, 60] and may need to be constantly evaluated to ensure anonymity [21]. Lastly, it is important to note that GDPR considers the act of *anonymizing personal data* as a data processing activity in itself [7]. So, the controller must ensure that they have a legitimate purpose or a legal basis to perform anonymization and it cannot be used as a tool to evade the applicability of GDPR. A case in point: in 2020, the Greek regulators reprimanded the insurance company Axa for anonymizing its customer data so that they could retain it beyond the date of expiry or date of RTBF request [28].

3. Keeping personal data untagged

Tagging personal data with attributes such as its purpose, time-to-live, associated person, objections to its use, etc., makes it possible to comply with people exercising their data rights. However, data tagging is a resource-intensive operation, both computationally and on the storage system [63]. Thus, there is a tendency to avoid it or at least, delay it to a point when data subjects actually exercise their rights. For example, consider Clearview AI—a company that collects images of people's faces from the public Internet and social media to build an online global database, which is then used to offer facial-recognition-as-a-service to law enforcement and private organizations. When EU citizens approached Clearview AI to exercise RTBF, they were instructed to provide photographs instead of just their identities, so that their photos could be matched in the database. Several regulators [20, 29, 54] found this practice to be an impediment to people exercising their RTBF rights, among other things, and issued penalties of over €40M.

4. Mismanaging the depth of deletion

$\mathscr{G}17$ does not specify how deep should a deletion be percolated, instead leaving its interpretation to data controllers and regulators. So, simply extending the existing delete workflows to RTBF may lead to non-compliant behavior. At one extreme, Brico Prive, a French online retailer, had the practice of simply deactivating the user handles upon receiving RTBF requests while still retaining the user data in the database. The French regulators deemed this an *incomplete deletion* and fined them €500K [19]. At the other end, consider Google Search's implementation of RTBF. When they determine a link has to be delisted, they not only delete it from their databases but also propagate the deletion request to the original website that published the content. The Swedish regulators fined Google for this practice €7M [11] citing that Google had no legal basis for *propagating deletion* requests externally (which would violate user's privacy).

5. Logging without checks and bounds

Logs are relatively inexpensive to generate and to store, and have been used for a variety of purposes including debugging and traceability, post-hoc analysis and audit, and as reliability primitives. Since GDPR allows (via $\mathscr{G}24$ and $\mathscr{G}30$) the use of computer logs as evidence in establishing compliance, organizations have tended to *play it safe* in storing any and all forms of logs for possible future use. However, logging can conflict with the spirit of RTBF. In 2022, the Belgian regulators reprimanded a controller for storing excessive information about RTBF requests in their logs. Given that GDPR defines the act of logging as a data processing activity in itself, care should be taken (i) to not store any RTBF deleted data in logs, (ii) to follow the principle of data minimization i.e., do not store anything that is not essential, and (iii) to set a date for expiring old logs. Any logging system that ignores these requirements risks violating RTBF.

6. Employing excessive verification

In contrast to other GDPR rights, RTBF, by definition, is for single and definitive use. That is, once RTBF is exercised successfully (i.e., a dataset has been erased), no other right could be exercised on the same dataset. This has prompted some organizations to employ stricter and excessive forms of user verification, in turn placing an undue burden on those exercising RTBF. For instance, the Irish regulators issued reprimands to Groupon in 2020 [24] and to Twitter in 2022 [25] for requiring national photo IDs to exercise RTBF but allowing people create accounts without such IDs.

Scope and limitations. The anti-patterns presented here should be treated as starting points towards making systems adapt better to RTBF. They should not be considered *exhaustive* (i.e., it may be possible to identify additional anti-patterns by analyzing the

Table 2: Exploring RTBF tradeoff with compliance profiles

	User interface	Response time	Policy resolution	Policy depth	Data identification	Deletion API	Deletion latency
Cost-optimal	native	30 days	automated	codified	ML-based	built-in	–
Perf-optimal	–	–	–	–	programmatic	built-in	lazy
Risk-optimal	multi-modal	ASAP per-request	manual	multi-regulatory	pre-tag	custom	real-time

corpora with a different perspective) or *prescriptive* (i.e., there is no guarantee that having these in your systems will incur a penalty nor that avoiding them will be enough to avoid a violation). Finally, anti-patterns are distinct from dark patterns [12, 14, 38, 47, 48] which are techniques and practices aimed at deceiving the users and manipulating them into taking certain actions. In contrast, anti-patterns do not have any malicious intent built-in; instead, they emerge by looking differently at principles and practices that have served as good patterns elsewhere.

3.4 RTBF Tradeoffs

We wind up our design section with a discussion on tradeoffs. Tradeoffs emerge naturally in RTBF because of the broad interpretability of the law. However, there is a minimum bar to be met: any compliant system must support the four basic functionalities of RTBF (as detailed in Section-3.1) and must avoid non-compliant behavior highlighted by prior enforcements (in Section-3.2). This still leaves us with a broad set of design and operational choices. Our goal here is to methodically evaluate this space and define compliance profiles that optimize for given metrics.

Compliance profiles. We choose performance, cost, and risk as the three foundational metrics to optimize on. We define *performance-optimal* to be the choice that maximizes the system's performance, while still meeting the minimal level of compliance. In contrast, *cost-optimal* would be the choice that achieves the lowest aggregate cost of designing and operating the system, while still meeting the performance requirements and being minimally compliant. Finally, *risk-optimal* is the choice that offers the lowest probability of RTBF violation while still maintaining the minimally required performance levels. To keep our analysis agnostic of any particular application, we treat our metrics as abstract entities and do not assign numerical values. Table 2 lists these compliance profiles.

Design alternatives and tradeoffs. We elaborate on seven design aspects that allow multiple valid ways to implement RTBF support. First, the *user interface*. By definition, all computing systems will have a user interface, and for modern systems, these tend to be web interface and/or mobile app. So, the cost-optimal option would be to add RTBF support on this native interface. On the other hand, offering RTBF support using multiple modalities (say email, telephone, postal mail, etc., in addition to native interfaces) will make it easier for a broad set of people to exercise RTBF, and thus the risk-optimal choice. Second, the *response time*. $\mathscr{G}$12(3) requires that all data right requests should be responded to within 30 days. So, the cost-optimal option is to process RTBF requests in batches once every 30 days. If one wants to be risk-optimal, then RTBF requests could be processed as soon as they arrive, and responded to on an individual basis without waiting for 30 days. Next two parameters concern *policy resolution*. The cost-optimal option would be to automate the process of policy handling by having a set of programmatic rules that determine the RTBF decision. While this approach saves on manual labor (and also speeds up the decision-making process), it is impossible to capture all the exemptions arising out of RTBF's dependency on other GDPR articles as well as the influence of non-GDPR laws on RTBF (for e.g., children's rights, domestic abuse laws, etc). So, the least risky approach, as practiced by Google Search [9], is to have an extensive policy framework and to allow qualified humans to adjudicate on RTBF requests on individual basis. Thus far, we have ignored providing options for performance-optimal version since the first four design aspects do not interfere with the runtime performance of the system.

The fifth parameter is *data identification* i.e., how to identify all the data items that match an RTBF request. The performance-optimal choice would be to programmatically identify all the data and then invoke deletion APIs on them. However, this may not always be possible since it requires data to be structured, pre-labeled, and all data storage systems to be fully connected. So, for organizations with large quantities of unstructured personal data collected over years, this may not be feasible. Instead, a cost-optimal approach would be to employ AI/ML services that identify data based on heuristics. For example, Amazon Macie [4] provides one such service. If the number of RTBF requests is small enough, then AI/ML based approaches will be cost-optimal. However, the risk-optimal strategy would be do pre-tag the entire corpora of personal data either at the time of ingestion or offline using human labelers. Spending the time and effort on labeling results in accuracy levels that are not realistic in current generation AI/ML systems.

The last two aspects concern the actual act of deletion. All software systems and data management systems must provide APIs to delete a given data (because if they don't, they cannot be used in a GDPR-compliant system). Using these native APIs will likely result in both performance- and cost-optimal solutions. However, deletions implemented by current generation systems may not be thorough or timely [27, 44, 61]. To remedy this, a cleansing delete could be implemented either by modifying the existing software systems or by using external libraries. Such cleansing deletes would clean up the data from all the internal data structures; memory,

cache and file subsystems; logs, snapshots, and replicas; and percolate the request to underlying hardware–thereby, providing the risk-optimal variant. Finally, the *deletion latency*. Though GDPR has set a strict upper bound on the RTBF response time, regulators have shown considerable leeway for deletion latency (as discussed in Section-3.2). So, a performance-optimal solution is to perform lazy deletions (i.e., mark the data as deleted in real-time, but perform the actual removal of the data asynchronously when the system load is low or periodically). In contrast, a risk-optimal approach would be to immediately trigger deletion since retaining any to-be deleted data exposes organizations to additional risks. For instance, if that date gets breached before the deletion cycle is completes.

Scope and limitations. First off, the set of design parameters presented in Table 2 are not exhaustive. Instead, it highlights only those aspects where the choices diverge for different optimization metrics. For instance, we do not include data subject verification, even though it is an important design consideration for RTBF systems. This is because there is one good way to do it, and alternative options may violate enforcement precedents. Second, compliance is contextual i.e., what a system must do to be compliant is dependent on a variety of factors such as type and volume of personal data, nature of processing, security measures employed, size of the organization, among others. Thus, compliance profiles presented here will not be universally applicable and may need to be adjusted for a given controller.

4 RTBF-CAPABLE ELASTICSEARCH

To ground our work, we select search engines as the target application. This is a natural choice since search engines were the first software system to be subject to RTBF (more details in Section-2.1). Importance of RTBF in search engines is emphasized by the fact that EDPB published a dedicated guidance on this matter in 2020 [32]. Our analysis and evaluation in the rest of the section are based on Elasticsearch [5], an open-source search engine that is amongst the most widely deployed in the world.

4.1 Is Elasticsearch RTBF Capable?

Elasticsearch is a distributed search engine that can support search analytics in near real-time at petabyte scale. Given its prominence in the data analytics ecosystem, several prior work have analyzed its GDPR readiness [6, 66]. However, in these analyses, support for RTBF has been simply reduced to *whether Elasticsearch provides APIs to delete data*. We aim to explore this question in more depth and with nuance, based on the design considerations of Section-3. Since Elasticsearch is primarily an application and data management system, our exploration will focus on these (while covering any applicable aspects of UI and policy layers). Below, we investigate Elasticsearch's RTBF capability along two axes: what RTBF tasks can it perform and how well does it perform them:

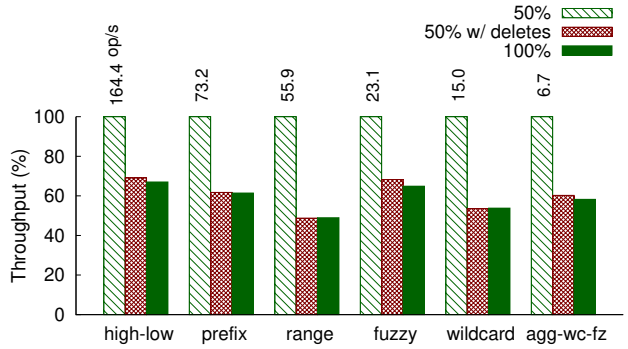
- **Deletion APIs.** Elasticsearch offers two direct options: the RESTful method `DELETE` and the API `_delete_by_query`. The former can be used to delete a document (equivalent of tuples in DBMS) or an index (equivalent of an entire database in DBMS), while the latter is useful for deleting a set of documents that match a query. Elasticsearch also provides a mechanism to remove contents within a document (equivalent of setting a field to null in DBMS) through the `_update_by_query` API. Finally, Elasticsearch used to support a `time-to-live` field for each document, which ensured that a given document is automatically deleted after the specified time, but this has been deprecated since version 5 (circa June 2018).
- **Deletion granularity.** It is straightforward to see that Elasticsearch's built-in deletion APIs and methods (discussed above) allow for data to be deleted at all granularities: individual key-value pairs and documents, groups of key-value pairs and documents that match a search criteria, and the entire indexes.
- **Deletion latency.** Elasticsearch takes a lazy approach to deletion i.e., all deletion APIs simply *mark the data as deleted* without actually erasing them from data structures and files; instead, the actual deletion is carried out at a later time depending on the runtime state of the system and user-defined configuration parameters (we elaborate on this in Section-4.3). While this approach helps Elasticsearch be highly performant, it makes it hard to determine the latency of deletion operations.
- **Deletion type.** Elasticsearch's deletion can be best categorized as *shallow*. Deletion APIs make the deleted data unavailable for the application, and eventually remove them from the internal data structures. However, they do not attempt to delete it from all the underlying subsystems which may contain the deleted data. For instance, query cache, translog, and snapshots, to name a few (we elaborate on this in Section-4.3).

In summary, Elasticsearch can support RTBF under cost- and performance-optimal configurations. However, we identify two aspects of RTBF where Elasticsearch's capabilities are lacking: cleansing deletion and deletion at scale. These make Elasticsearch unsuitable for risk-optimal configurations.

4.2 Deletion at Scale

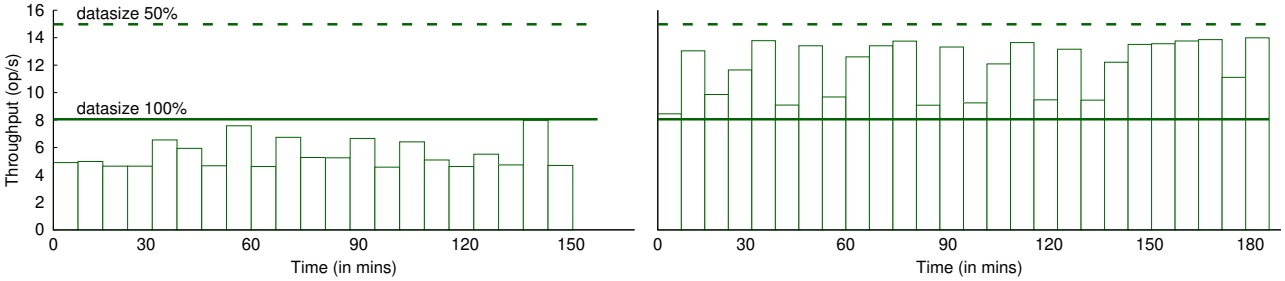
Scalability is one of the core aspects of Elasticsearch design, so we investigate how deletions of scale are handled. We do so under two configurations: bulk deletion (i.e., issuing all the delete requests at once) and streaming deletion (i.e., issuing delete requests based on a Poisson distribution). Bulk deletion represents the practice of batching all the delete requests that have arrived in a window of time, whereas streaming deletion represents a risk-averse approach of completing the deletions as and when they arrive.

Experimental setup. We perform our experiments on Chameleon Cloud [42]. Elasticsearch is run on a dedicated Dell PowerEdge R6525 server with 64-core AMD EPYC 2.45GHz processor, 256MB cache, 256GB memory, 480GB SSD storage, and Gigabit Ethernet. We use Elasticsearch version 8.10 (released Sep 2023) and run it in single shard, zero replica configuration. For the bulk delete experiment, we disable frequent merges by setting `deletes_pct_allowed` to 50 and `floor_segment` to 5GB; whereas in streaming delete case, we revert these to 5% and 1MB respectively to encourage frequent merges. We benchmark Elasticsearch's performance using Rally (version 2.10, released in Nov 2023) [52]. In particular, we use the StackOverflow track, which consists of 36 million documents, each of which is a question and answers post from StackOverflow, adding



Challenge	Workload description
high-low	boolean AND of high occurrence and low occurrence queries
prefix	search based on prefix within a field
range	search based on a date range
fuzzy	search based on approx. string matching
wildcard	search based on wildcard patterns
agg-wc-fz	wildcard based fuzzy search followed by aggregation

Figure 4: Impact of large-scale bulk deletion. After bulk deletion of 50% data, Elasticsearch throughput remains 30-50% below the expected level across a diverse set of Rally challenges.



(a) Day 1 (datasize close to 100%)

(b) Day 7 (datasize close to 50%)

Figure 5: Impact of long-term streaming deletion. When exposed to streaming deletes, Elasticsearch exhibits up to 40% variation in throughput, while proportionally improving its performance due to repeated merges.

up to a total repository size of 33GB. Table in Figure 4 lists six challenges (or workloads) representing a broad mix of query complexity that we use in our benchmarking. We configure a benchmark run as 200 iterations of warm up followed by 500 iterations of challenge queries.

For this experiment, we define deletion at scale as deleting 50% of the data. To understand how Elasticsearch handles it, we establish two baselines: its performance at 100% datasize (33GB) and its performance at 50% datasize (16.5 GB). Intuitively, we expect the throughput at 50% datasize to be nearly twice that of 100% datasize (since we are not scaling up our compute resources for the additional datasize). We experimentally measure and plot these in Figure 4 as green striped bars and green solid bars respectively. On the Y-axis, we represent the mean throughput (normalized to the 50% case) and on the X-axis, we list the Rally challenges. Now, to measure the impact of bulk delete: we start Elasticsearch at 100% datasize, issue requests to delete 50% data via bulk delete API, wait till we get success response from Elasticsearch, and then run the benchmark challenges. We show our findings in Figure 4 using red checkered bars. We see that after bulk deletion, performance remains roughly at the same level as that of 100% datasize. In other words, Elasticsearch operates at 30% to 50% below the expected levels after a large-scale deletion. This disparity stems from Elasticsearch's design decision to retain the deleted data in its data structures by simply marking them as *expired*. So, even when the actual datasize reduces by 50%, the internal data structures to not shrink, in turn making the search algorithms work twice as hard only to discard half of their results later. To remedy this situation, a `forcemerge` with `expunge-deletes` option has to be triggered—an IO- and compute-intensive process that disrupts Elasticsearch's steady performance for long durations of time (for example, at this scale, the merge takes ~23 minutes to complete).

Next, we present the streaming delete case. Unlike the bulk delete case, where we had disabled the shard merge process during the benchmark runs, here we revert Elasticsearch's behavior to its default configuration. Figure 5 plots the instantaneous throughput (averaged over 7 minutes, on the Y-axis) and time since starting the benchmark run (on the X-axis) for a representative Rally challenge (in this case, wildcard). Similar to the previous experiment, we plot two baselines: 50% datasize and 100% datasize, using dotted green line and solid green line respectively. Expectedly, both of them produce a consistent throughput for the entirely of the benchmark run, and the former is 87% better than the latter. To measure the impact of streaming delete: we start Elasticsearch at 100% datasize, start the benchmark run, and then continually issue Poisson-modeled delete requests at an average of 30 documents per second (which is equivalent to deleting 50% of data over the span of a week). In Figure 5a, we show the performance on day-1,

Table 3: Exploring the component-level data storage in Elasticsearch

	Data structures	Elastic caches	Translog	Snapshots	Event log	Replica shard
Purpose	performance	performance	fault tolerance	fault tolerance	monitoring	availability
Storage location	memory, disk	memory, cache	disk	disk	disk	external server
Retention beyond deletion	until merge	depends on system load	until flush	no limit	no limit	same as primary
API to delete select data	yes but timing is not guaranteed	no	no	no	no	no

when streaming deletes start rolling in. The observed throughput is not only below the solid green bar, but also shows ~40% variation from the expected value. This is due to Elasticsearch's handling of additional delete request as well as due to the merge and flush process being triggered during the benchmark run. At this stage of streaming delete, the performance looks worse than the bulk delete case. However, streaming delete paints a different picture, when we fast forward to day 7 (in Figure 5b), at which point more than 85% of deletions are already completed. As the benefit of these deletions is baked into Elasticsearch state (via repeated merge operations), we see that the throughput has improved in proportion to the deleted data. Of course, it still shows abrupt variations as new deletion requests keep coming in and as merge/flush operations kick in. Finally, we have verified that once all the deletions are completed, the throughput stabilizes at the dotted green bar level.

Summary. Our experiments demonstrate a tradeoff between bulk delete and streaming delete approaches. The former produces a consistent albeit lower throughput, while the latter exhibits a jittery but constantly improving throughput.

4.3 Cleansing Delete

We describe a deletion operation as *timed* if it has a notion of time associated with it. For example, TTL-based deletion, where data would be deleted at a predetermined time; or real-time deletion, where data would be deleted synchronously in real-time. Similarly, we characterize a deletion to be *thorough* if all copies of the given data are erased from all parts of the system. This is distinct from sanitizing deletes [41] where data has to be irrevocably erased from the physical hardware; instead, a thorough delete simply goes through all the subsystems that may contain the given data and issues a delete request on them. Combining these two properties, we define *cleansing delete* as a delete operation that is both thorough and timed.

Cleansing deletes may seem unnecessary or excessive–after all, neither the language of the law nor the enforcement of the law have explicitly required this. However, if one were to consider deletion as a first-class operation, then optimizing for its speed and quality would seem like a natural choice. We think of this as extending the notion of *clean up after yourself* to data management systems. In fact, cleansing deletes are not without a precedent. Consider, Google cloud's deletion guarantee [3] that all copies of the data would be deleted from all underlying subsystems within 180 days of requesting it. In contrast, Elasticsearch's deletion is neither thorough nor timed. Our goal is to investigate why Elasticsearch's deletion is not cleansing, how to fix it, and how it impacts Elasticsearch performance.

Elasticsearch's data footprint. Table 3 shows the main components of Elasticsearch in which data gets stored during the create, read, and update operations. First, the internal data structures. Both Elasticsearch and Lucene (the search engine library on which Elasticsearch is built) employ a variety of data structures to efficiently index, query, and manage documents. These include inverted index, k-dimensional B-Trees, document-value structures, deleted document index, among others. Depending on the size of data, access patterns, available hardware resources, etc, these data structures will be stored in either in memory or on disk (or split between the two). While Elasticsearch delete APIs allow fine-grained deletions, they simply mark the deleted item with a `tombstone` but leave them as is in the data structures. So, the only way to guarantee a complete and immediate removal of deleted data is to trigger a `forcemerge`—a practice not recommended by Elasticsearch [33].

Second is the caching system used by Elasticsearch to speedup search performance which includes the page cache, request cache, and query cache [59]. The page cache is typically managed by the OS and lacks the semantic understanding of the data being cached. The request cache stores full responses to complex queries (such as aggregations) that are time-consuming to be run again, while the query cache uses heuristics to identify portions of the query responses that may be useful for other queries and stores them selectively. Neither Elasticsearch nor OS provide any APIs to evict select data items, even the deleted ones; so, the only option to remove any item is to clear the entire cache.

The next two components, transaction log and snapshots, bring fault tolerance to Elasticsearch. Translog is a record of all modifications to Elasticsearch (such as inserts, updates, and deletes) that have been accepted but not yet been committed to the Lucene index on the disk. The idea is to flush changes to the Lucene index in batches, so the overhead is amortized. However, to make this scheme crash consistent (i.e., retain modifications after a process crash or hardware failure), Translog itself has to be continually saved to the disk. Issuing a delete request on a data item that is present in Translog will not remove it from Translog; instead, it

will continue to be there until Translog becomes large enough to be flushed (controlled by the parameter `flush_threshold_size`). On the other hand, snapshot is a full backup of an Elasticsearch instance that can be used to transfer Elasticsearch between servers and to restore after a hardware failure. These exist as immutable files on the disk. Unfortunately, there are no APIs to remove select data items from a snapshot. The only way to achieve this would be to restore a snapshot, perform the deletions, and then create a new snapshot in its place.

The fifth element is the logging mechanism. While Elasticsearch generates several types of logs during its operation, two of them can contain actual data: Elasticsearch.log when set to the level `trace` and slowlog. By default, these are stored on the disk and are automatically compressed with passage of time. Elasticsearch neither provides an API to search logs that contain a given data item nor offer mechanisms to selectively delete items. However, since log files are in human readable ASCII, standard file I/O operations could be used to accomplish these tasks.

The final component is the replica shard. Elasticsearch employs a primary-backup model to improve availability. All indexing operations of Elasticsearch are directed to the primary, which is then responsible for pushing any change in its state to all the replicas. On the other hand, read requests can be sent to either primary or one of the replicas, depending on the system load. Thus, replica nodes will have their own data structures, caches, translog, and event logs as discussed in Table 3. However, Elasticsearch does not provide any APIs directly manipulate these on the replicas, instead requiring all changes to be driven by the primary (in order to keep them in sync). It must be noted that the delay in propagating the changes from primary to replicas is negligible since these are handled synchronously during API calls.

Cleansing delete and benchmarking. Next, we share our effort at introducing cleansing delete for Elasticsearch. We choose to implement this external to Elasticsearch (i.e., using its existing APIs and OS support) as opposed to redesigning the internals of Elasticsearch. Since our goal is to demonstrate the feasibility of cleansing delete and estimate the resulting overhead, this approach is sufficient (though we acknowledge that redesigning Elasticsearch would lead to a more optimal solution). Here are our five steps of cleansing: (i) for data structures, we issue `forcemerge` with `expunge-deletes` option set, (ii) for cache system, we invoke the OS command `drop_caches` and Elasticsearch's index-level clear cache API, (iii) for Translog, we issue Elasticsearch's index-level flush API, (iv) for snapshot, we restore it on a new server, issue delete requests on the select data, flush and forcemerge the index, and then create a new snapshot to replace the old one, (v) finally, for event logs, we iterate through all the files in the log directories and edit out the select data. We implement all these routines in ∼ 450 lines of Python code.

To benchmark our cleansing delete, we use the same experimental setup as in Section-4.2. Forcemerge on index takes O(milliseconds) to O(minutes), while flush runs in O(milliseconds) to O(seconds). Elasticsearch can continue to run workloads, albeit at a reduced throughput, while these two operations are going on. Clearing the cache is near instantaneous, taking just a few milliseconds, but impacts the performance of Elasticsearch significantly until the caches become hot again. Restoring and creating a snapshot on the full StackOverflow workload takes ∼12 minutes each in our setup, while event log culling runs in O(seconds). So, if we delete a single document and then issue a cleansing delete in our setup, the whole process takes about 24 minutes to complete and is bottlenecked by snapshot cleansing. On the other hand, if we delete 50% of documents (as in Section-4.2) and then trigger cleansing deletion, the forcemerge operation takes ∼23 minutes, taking the total time to 45 minutes.

5 CONCLUSION

Ever since its introduction, RTBF has triggered vigorous debates in our society–from being hailed it as a counterbalance to the aggressive data practices of the 21st century to being criticized as a means to rewrite history, weaken the freedom of expression, and enable censorship. This work is an attempt to bring clarity about implementing RTBF from a computing and data management perspective. We believe that our principled approach to understanding the law and enforcement of RTBF, and then translating them to implementable actions in computing systems, will bridge the disconnect between legal and computing domains.

Acknowledgment. Any opinions, findings, or recommendations expressed herein are those of the authors; these should neither be interpreted as legal advice nor as reflective of the views of their host institution.

REFERENCES

[1] 2012. Charter of Fundamental Rights of the European Union. *Official Journal of the European Union* 55, 391-407 (2012).
[2] 2016. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46. *Official Journal of the European Union* 59, 1-88 (2016).
[3] 2018. Data Deletion on Google Cloud Platform. https://cloud.google.com/security/deletion/.
[4] 2023. Amazon Macie: Discover and protect your sensitive data at scale. https://aws.amazon.com/macie/.
[5] 2023. ElasticSearch. https://github.com/elastic/elasticsearch.
[6] 2023. GDPR Compliance and The Elastic Stack. https://www.elastic.co/pdf/white-paper-elastic-gdpr-compliance-and-the-elastic-stack.pdf.
[7] AEPD and EDPS. 2021. AEPD-EDPS joint paper on 10 misunderstandings related to anonymisation. https://edps.europa.eu/system/files/2021-04/21-04-27_aepd-edps_anonymisation_en_5.pdf.
[8] Steven Bauer and Nissanka B Priyantha. 2001. Secure data deletion for Linux file systems. In *USENIX Security*.
[9] Theo Bertram, Elie Bursztein, Stephanie Caro, Hubert Chao, Rutledge Chin Feman, Peter Fleischer, Albin Gustafsson, Jess Hemerly, Chris Hibbert, Luca Invernizzi, et al. 2019. Five years of the right to be forgotten. In *ACM CCS*.
[10] European Data Protection Board. 2019. The Danish Data Protection Agency proposes a DKK 1.2 million fine for Danish taxi company. https://edpb.europa.eu/news/national-news/2019/danish-data-protection-agency-proposes-dkk-12-million-fine-danish-taxi_en.
[11] European Data Protection Board. 2020. The Swedish Data Protection Authority imposes administrative fine on Google. https://edpb.europa.eu/news/national-news/2020/swedish-data-protection-authority-imposes-administrative-fine-google_en.
[12] Christoph Bösch, Benjamin Erb, Frank Kargl, Henning Kopp, and Stefan Pfattheicher. 2016. Tales from the dark side: privacy dark strategies and privacy dark patterns. *PoPETS* 2016, 4 (2016), 237–254.
[13] Lucas Bourtoule, Varun Chandrasekaran, Christopher A Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. 2021. Machine unlearning. In *IEEE Symposium on Security and Privacy*.
[14] H Brignull, M Leiser, C Santos, and K Doshi. 2023. Deceptive patterns – user interfaces designed to trick you. https://www.deceptive.design/

[15] William Brown, Raphael Malveau, Hays McCormick, and Thomas Mowbray. 1998. *AntiPatterns: refactoring software, architectures, and projects in crisis*. John Wiley & Sons, Inc.

[16] Yinzhi Cao and Junfeng Yang. 2015. Towards making systems forget with machine unlearning. In *IEEE symposium on security and privacy*.

[17] Grand Chamber. 2014. Google Spain SL and Google Inc. v Agencia Española de Protección de Datos (AEPD) and Mario Costeja González. https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:62012CJ0131. *The Court of Justice of the European Union - Reports of Cases* (2014).

[18] Min Chen, Zhikun Zhang, Tianhao Wang, Michael Backes, Mathias Humbert, and Yang Zhang. 2022. Graph unlearning. In *ACM CCS*.

[19] CNIL. 2021. Deliberation SAN-2021-008. https://www.legifrance.gouv.fr/cnil/id/CNILTEXT000043668709.

[20] CNIL. 2022. Facial recognition: 20 million euros penalty against CLEARVIEW AI. In *Restricted Committee Deliberation No. SAN-2022-019*. (Oct 20 2022).

[21] Aloni Cohen and Kobbi Nissim. 2020. Towards formalizing the GDPR's notion of singling out. *PNAS* 117, 15 (2020), 8344–8352.

[22] Katriel Cohn-Gordon, Georgios Damaskinos, Divino Neto, Joshi Cordova, Benoît Reitz, Benjamin Strahs, Daniel Obenshain, Paul Pearce, and Ioannis Papagiannis. 2020. DELF: safeguarding deletion correctness in online social networks. In *USENIX Security*.

[23] Irish Data Protection Commission. 2019. Guidance on Anonymisation and Pseudonymisation. https://www.dataprotection.ie/sites/default/files/uploads/2019-06/190614%20Anonymisation%20andi%20Pseudonymisation.pdf.

[24] Ireland Data Protection Commission. 2020. Groupon International Limited - December 2020. https://dataprotection.ie/en/dpc-guidance/law/decisions/groupon-december-2020.

[25] Ireland Data Protection Commission. 2022. Twitter International Company - April 2022. https://dataprotection.ie/en/resources/law/decisions/twitter-international-company-april-2022.

[26] Sophie Curtis and Alice Philipson. 2014. Wikipedia founder: EU's Right to be Forgotten is deeply immoral. In *The Telegraph*. (Aug 6 2014).

[27] Siying Dong, Mark Callaghan, Leonidas Galanis, Dhruba Borthakur, Tony Savor, and Michael Strum. 2017. Optimizing Space Amplification in RocksDB.. In *CIDR*, Vol. 3. 3.

[28] Hellenic DPA. 2020. Complaint by a prospective insured against an insurance company for failure to satisfy the right to deletion. https://www.dpa.gr/el/enimerwtiko/prakseisArxis/kataggelia-ypopsifioy-asfalismenoy-kata-asfalistikis-etaireias-gia-mi.

[29] Hellenic DPA. 2022. Hellenic DPA fines Clearview AI 20 million euros. In *EDPB News*. (Jul 20 2022).

[30] David Drummond. 2014. We need to talk about the right to be forgotten. In *The Guardian*. (Jul 10 2014).

[31] Alan M Dunn, Michael Z Lee, Suman Jana, Sangman Kim, Mark Silberstein, Yuanzhong Xu, Vitaly Shmatikov, and Emmett Witchel. 2012. Eternal sunshine of the spotless machine: Protecting privacy with ephemeral channels. In *USENIX OSDI*.

[32] EDPB. 2020. Guidelines 5/2019 on the criteria of the Right to be Forgotten in the search engines cases under the GDPR. https://edpb.europa.eu/sites/default/files/consultation/edpb_guidelines_201905_rtbfsearchengines_forpublicconsultation.pdf. (Jul 7 2020).

[33] Elasticsearch. 2024. Force merge API. In *Elastic Guide*. https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-forcemerge.html. (Jan 17 2024).

[34] Wikimedia Foundation. 2023. Transparency Reports. https://wikimediafoundation.org/about/transparency/.

[35] Sanjam Garg, Shafi Goldwasser, and Prashant Nalini Vasudevan. 2020. Formalizing data deletion in the context of the right to be forgotten. In *EUROCRYPT*.

[36] Samuel Gibbs. 2014. Larry Page: right to be forgotten could empower government repression. In *The Guardian*. (May 30 2014).

[37] Antonio Ginart, Melody Guan, Gregory Valiant, and James Y Zou. 2019. Making ai forget you: Data deletion in machine learning. *Advances in neural information processing systems (NeurIPS)* 32 (2019).

[38] Colin Gray, Yubo Kou, Bryan Battles, Joseph Hoggatt, and Austin Toombs. 2018. The dark (patterns) side of UX design. In *Proceedings of the 2018 CHI conference on human factors in computing systems*.

[39] Hana Habib, Sarah Pearman, Jiamin Wang, Yixin Zou, Alessandro Acquisti, Lorrie Faith Cranor, Norman Sadeh, and Florian Schaub. 2020. " It's a scavenger hunt": Usability of Websites' Opt-Out and Data Deletion Choices. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*.

[40] Hana Habib, Yixin Zou, Aditi Jannu, Neha Sridhar, Chelse Swoopes, Alessandro Acquisti, Lorrie Faith Cranor, Norman Sadeh, and Florian Schaub. 2019. An empirical analysis of data deletion and opt-out choices on 150 websites. In *Proceedings of the 15th Symposium on Usable Privacy and Security (SOUPS 2019)*.

[41] Gordon F Hughes, Tom Coughlin, and Daniel M Commins. 2009. Disposal of disk and tape data by secure sanitization. *IEEE Security & Privacy* 7, 4 (2009).

[42] Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S. Gunawi, Cody Hammock, Joe Mambretti, Alexander Barnes, François Halbach, Alex Rocha, and Joe Stubbs. 2020. Lessons Learned from the Chameleon Testbed. In *USENIX ATC*.

[43] Andrew Koenig. 1995. Patterns and Antipatterns. *Journal of Object-Oriented Programming* 8, 1 (1995), 46–48.

[44] Andrew Lamb, Matt Fuller, Ramakrishna Varadarajan, Nga Tran, Ben Vandiver, Lyric Doshi, and Chuck Bear. 2012. The Vertica Analytic Database: C-Store 7 Years Later. *PVLDB* 5, 12 (2012).

[45] CMS Law. 2023. GDPR Enforcement Tracker. https://www.enforcementtracker.com/.

[46] Yi Liu, Lei Xu, Xingliang Yuan, Cong Wang, and Bo Li. 2022. The Right to be Forgotten in Federated Learning: An Efficient Realization with Rapid Retraining. In *IEEE INFOCOM*.

[47] Arunesh Mathur, Gunes Acar, Michael J Friedman, Eli Lucherini, Jonathan Mayer, Marshini Chetty, and Arvind Narayanan. 2019. Dark patterns at scale: Findings from a crawl of 11K shopping websites. *Proceedings of the ACM on Human-Computer Interaction* 3, CSCW (2019), 1–32.

[48] Arunesh Mathur, Mihir Kshirsagar, and Jonathan Mayer. 2021. What makes a dark pattern... dark? design attributes, normative considerations, and measurement methods. In *Proceedings of the 2021 CHI conference on human factors in computing systems*.

[49] Viktor Mayer-Schönberger. 2011. *Delete: The virtue of forgetting in the digital age*. Princeton University Press.

[50] Microsoft. 2023. Right to be forgotten Requests. https://www.microsoft.com/en-us/corporate-responsibility/right-to-be-forgotten-removal-requests-report.

[51] Mohsen Minaei, Mainack Mondal, and Aniket Kate. 2022. Empirical Understanding of Deletion Privacy: Experiences, Expectations, and Measures. In *USENIX Security*.

[52] Daniel Mitterdorfer. 2016. Announcing Rally: Our benchmarking tool for Elasticsearch. In *Elastic Blog*. https://www.elastic.co/blog/announcing-rally-benchmarking-for-elasticsearch. (Apr 19 2016).

[53] Arvind Narayanan and Vitaly Shmatikov. 2008. Robust de-anonymization of large sparse datasets. In *IEEE Symposium on Security and Privacy*. 111–125.

[54] UK Information Commissioner's Office. 2022. ICO fines facial recognition database company Clearview AI Inc more than 7.5M GBP and orders UK data to be deleted. In *Monetary Penalty Notices*. (May 22 2022).

[55] Data Protection Working Party. 2014. Opinion 05-2014 on Anonymisation Techniques. https://ec.europa.eu/justice/article-29/documentation/opinion-recommendation/files/2014/wp216_en.pdf.

[56] Eugenia Politou, Alexandra Michota, Efthimios Alepis, Matthias Pocs, and Constantinos Patsakis. 2018. Backups and the right to be forgotten in the GDPR: An uneasy relationship. *Computer Law & Security Review* 34, 6 (2018).

[57] Joel Reardon, David Basin, and Srdjan Capkun. 2013. Sok: Secure data deletion. In *IEEE symposium on security and privacy (OAKLAND)*.

[58] Joel Reardon, Hubert Ritzdorf, David Basin, and Srdjan Capkun. 2013. Secure data deletion from persistent media. In *ACM CCS*.

[59] Alexander Reelsen. 2021. Elasticsearch caching deep dive: Boosting query speed one cache at a time. In *Elastic Blog*. https://www.elastic.co/blog/elasticsearch-caching-deep-dive-boosting-query-speed-one-cache-at-a-time. (Mar 4 2021).

[60] Luc Rocher, Julien M Hendrickx, and Yves-Alexandre De Montjoye. 2019. Estimating the success of re-identifications in incomplete datasets using generative models. *Nature communications* 10, 1 (2019), 1–9.

[61] Subhadeep Sarkar, Tarikul Islam Papon, Dimitris Staratzis, and Manos Athanassoulis. 2020. Lethe: A tunable delete-aware LSM engine. In *ACM SIGMOD*.

[62] Max Schrems. 2023. GDPRhub. https://gdprhub.eu/.

[63] Supreeth Shastri, Vinay Banakar, Melissa Wasserman, Arun Kumar, and Vijay Chidambaram. 2020. Understanding and Benchmarking the Impact of GDPR on Database Systems. *PVLDB* 13, 7 (2020).

[64] Chen Sun, Evan Jacobs, Daniel Lehmann, Andrew Crouse, and Supreeth Shastri. 2023. GDPRxiv: Establishing the State of the Art in GDPR Enforcement. *Proceedings on Privacy Enhancing Technologies (POPETS)* 4 (2023).

[65] Kejsi Take, Kevin Gallagher, Andrea Forte, Damon McCoy, and Rachel Greenstadt. 2022. "it feels like whack-a-mole": User experiences of data removal from people search websites. *Proceedings on Privacy Enhancing Technologies* 3 (2022).

[66] Ana Paula Vazão, Leonel Santos, Rogério Luís de C Costa, and Carlos Rabadão. 2023. Implementing and evaluating a GDPR-compliant open-source SIEM solution. *Journal of Information Security and Applications* 75 (2023), 103509.

[67] James Q Whitman. 2003. The two western cultures of privacy: Dignity versus liberty. *Yale Law Journal* 113 (2003), 1151.

[68] Dawen Zhang, Pamela Finckenberg-Broman, Thong Hoang, Shidong Pan, Zhenchang Xing, Mark Staples, and Xiwei Xu. 2023. Right to be Forgotten in the Era of Large Language Models: Implications, Challenges, and Solutions. *arXiv preprint arXiv:2307.03941* (2023).

[69] Jonathan Zittrain. 2015. The right to be forgotten ruling leaves nagging doubts. In *The Financial Times*. (Jul 19 2015).