Parasol: Efficient Parallel Synthesis of Large Model Spaces

Clay Stevens
University of Nebraska-Lincoln
School of Computing
Lincoln, Nebraska, USA
clay.stevens@huskers.unl.edu

Hamid Bagheri
University of Nebraska-Lincoln
School of Computing
Lincoln, Nebraska, USA
bagheri@unl.edu

ABSTRACT

Formal analysis is an invaluable tool for software engineers, yet state-of-the-art formal analysis techniques suffer from well-known limitations in terms of scalability. In particular, some software design domains—such as tradeoff analysis and security analysis—require systematic exploration of potentially huge model spaces, which further exacerbates the problem. Despite this present and urgent challenge, few techniques exist to support the systematic exploration of large model spaces. This paper introduces Parasol, an approach and accompanying tool suite, to improve the scalability of large-scale formal model space exploration. Parasol presents a novel parallel model space synthesis approach, backed with unsupervised learning to automatically derive domain knowledge, guiding a balanced partitioning of the model space. This allows Parasol to synthesize the models in each partition in parallel, significantly reducing synthesis time and making large-scale systematic model space exploration for real-world systems more tractable. Our empirical results corroborate that Parasol substantially reduces (by 460% on average) the time required for model space synthesis, compared to state-of-the-art model space synthesis techniques relying on both incremental and parallel constraint solving technologies as well as competing, non-learning-based partitioning methods.

CCS CONCEPTS

• **Software and its engineering** → **Formal software verification**; • **Security and privacy** → *Logic and verification.*

KEYWORDS

formal analysis, bounded verification, tradespace analysis, parallel

ACM Reference Format:
Clay Stevens and Hamid Bagheri. 2022. Parasol: Efficient Parallel Synthesis of Large Model Spaces. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22), November 14–18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3540250.3549157

ESEC/FSE '22, November 14–18, 2022, Singapore, Singapore

ACM ISBN 978-1-4503-9413-0/22/11…$15.00
https://doi.org/10.1145/3540250.3549157

1 INTRODUCTION

Formal modeling of software systems has long been a hallmark of rigorous software engineering. The ability to systematically and formally analyze the properties and behavior of a system can greatly benefit the system's quality, security, and performance. Formal analysis techniques have been successfully used in a wide variety of applications, ranging from more theoretical uses like theorem proving [29] and bounded verification [22] to more practical applications such as configuration selection [33, 35] and self-adaptive systems [49]. Recent advances in the field have also lead to more widespread industry adoption [14, 37, 41, 52]. However, these techniques face well-known challenges with scalability when analyzing large-scale systems. As the number of variables in the formal specification of the system grows, the number of possible *models* of the specification grows exponentially–the so-called "curse of dimensionality". Thus, formal analysis techniques must search a vast *model space* to find models that satisfy all constraints in the specification.

In many application domains, the problem is further exacerbated by the need to not only find a *single* satisfying model, but to instead *explore* the entire model space to find all satisfying models. In the area of *tradeoff analysis*, for example, system designers must balance the needs of multiple stakeholders and conflicting objectives to select the best design for a system; this necessitates a systematic analysis of the tradeoffs among *all* possible designs. For large-scale systems, manual exploration of design variants will likely exclude possible design alternatives that would be otherwise optimal candidates, leading to a premature fixation on potentially non-optimal designs [21, 44, 53]. The development of efficient formal techniques to model and explore these design tradeoff spaces is therefore an active area of research [6, 12, 13, 17, 28, 34, 43, 44, 48, 56].

Similarly, in *security analysis*, analysts must explore large model spaces when identifying and addressing possible security threats to their systems. The growing popularity of consumer IoT systems increases the need for scalable systems to model and identify cyber-physical threats, which again necessitates a systematic exploration of a large and ever-growing model space of possible security risk models [1, 8, 19, 47]. In both of these areas, systematic model space exploration has been successfully applied within constrained sub-domains such as embedded systems or computer hardware design [31, 45]. However, using such techniques faces steep challenges when applied to software systems, where the models spaces are often colossal.

In fact, the major challenge limiting the application of systematic model space exploration to the software engineering domain is the problem of exhaustive *model space synthesis*. While recent researchers have presented systematic approaches which guarantee

a *complete* enumeration of the model space for a software system [6, 12], they do not address the *scalability* of model space synthesis. Exploring the model space for even modestly-sized software systems can quickly become intractable for the existing model space synthesis techniques, limiting the utility of existing tools for software engineers in practice.

To address the problem of scalable exploration of large model spaces, we present a novel approach and accompanying tool suite, dubbed Parasol, for **para**llel **s**ynthesis **o**f **l**arge model spaces. Parasol leverages unsupervised learning to effectively support systematic model space synthesis *in parallel*, by dividing the model space into smaller, non-overlapping partitions, which can then be synthesized concurrently. Parasol improves upon state-of-the-art model space synthesis techniques by addressing two of the factors currently limiting their scalability in practice. (1) First, in order to ensure synthesis of the entire model space, the model space synthesis problem must be incremented after synthesizing each model to exclude previously synthesized models and prevent duplicates. *This self-referencing reliance on earlier solutions to avoid duplication of effort forces the existing techniques to operate sequentially.* (2) Second, using existing approaches, additional constraints must be added to the model space synthesis problem to exclude each newly synthesized model. *As more constraints are added to the problem, the time required to synthesize each new model grows, ultimately requiring greatly increased time to synthesize each of the last models.*

Parasol overcomes both limitations, enabling efficient parallelization of the model space synthesis problem. Parasol first generates a bounded sample of the model space through a rigorous analysis of the system specification, and then clusters the models for that sample via unsupervised learning. The invariants for those clusters are then automatically derived and captured as formal partition definitions such that synthesis of the target model space can be performed in parallel. By partitioning the problem according to the clusters discovered in the sample, Parasol allows each parallel worker to synthesize a portion of the model space entirely independently of the other workers, improving the efficiency of the process while still avoiding synthesis of duplicate models. Also, Parasol's parallel synthesis mitigates the increased processing time due to large numbers of constraints by dividing them among multiple, smaller problems, reducing the impact of each additional constraint.

The results of our experimental evaluation over a diverse set of subject systems corroborate that Parasol greatly reduces the total time required for systematic model space synthesis compared to state-of-the-art approaches, while introducing very little overhead. Parasol provides an average speedup of 460% over state-of-the-art model space synthesis approaches, including the sampling overhead, which accounts for less than 7% of the total synthesis time.

To summarize, we make the following contributions:

- *Efficient, learning-driven parallel model space synthesis:* We introduce a novel parallel model space synthesis approach, backed with unsupervised learning to automatically derive domain knowledge, guiding a balanced partitioning of the model space, and thereby enabling efficient synthesis.
- *Parasol implementation:* We realize the presented approach in a tool, called Parasol, which we make available to the research and education community [5].
- *Experiments:* We present empirical evidence of the efficiency gains when synthesizing model spaces for real-world specifications adapted from prior work.

The following section presents necessary background to describe our technique and running example. Section 3 details the approach, and Section 4 describes our empirical evaluation. We discuss the results and validity of our experiments in Section 5, concluding with a review of related research and some remarks on future directions.

2 BACKGROUND AND MOTIVATION

To further motivate the research and illustrate our approach, we provide a running example of developing an efficient database design having to do with a systematic model space synthesis of the possible system-specific database design alternatives. Consider object-relational database mapping (ORM) tools, now provided in many popular software libraries (e.g., Hibernate [18]) and frameworks (e.g., Django [16]). They map object-oriented data models to relational database schemas for managing application data. These mappings employed by an ORM design tool significantly impact data storage and retrieval performance for the enclosing system.

Figure 1 shows three possible mappings for a partial object model of an e-commerce system (adapted from Lau and Czarnecki [30]) that allows *Customer*s to place *Order*s within the system, with an additional subclass defined for *Member* customers, who receive differential treatment. The full e-commerce system incorporates a large tradespace, with thousands of possible design alternatives, as shown in the scatter plot in Figure 1, where each grey circle on the scatter plot represents a unique, valid database design alternative. Larger data models will have an even larger tradespace. As the number of associations or inheritance relations in the domain model grows, the number of possible variants will grow exponentially; each relationship would multiply the total by the number of possible strategies that could be assigned to that relationship.

Picking the best database design and object-relational mapping often requires analyzing tradeoffs among candidates (e.g., query vs. update speed), necessitating systematic exploration of the entire model space; otherwise the designer may only consider suboptimal designs. However, the current state-of-the-practice for ORM design tools produces database designs based on a single-point strategy [7], considering no/limited non-functional properties and ignoring the performance ramifications for the system [16, 18, 24]. The star highlighted in Figure 1 denotes the point design solution produced by a state-of-the-practice ORM design tool. It is clear from the diagram that the database design generated by state-of-the-practice design tools is not among the Pareto-optimal designs, highlighted by triangles in Figure 1. The design produced by the state of practice—widely used every day by thousands of developers—is far away from Pareto optimal solutions because such design tools fail to consider the entire model space, focusing on a single solution. The challenge is that generating large numbers of complex variants is expensive. Thus, in order to enable the selection of (Pareto-)optimal design solutions, system designers require tools that can systematically and *scalably* generate the entire model space.

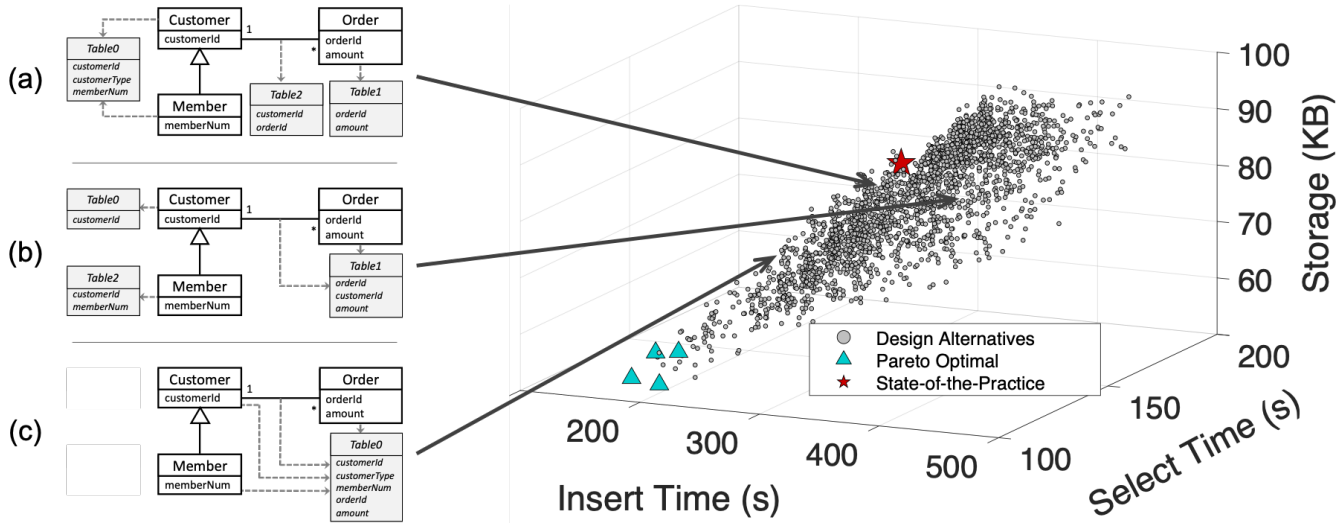


Figure 1: Example model space with quality attributes (i.e., tradespace) for database designs for an e-commerce system, comparing insert and select time and required storage space. (a)-(c) present partial object-relational mappings for three design alternatives, indicated in the scatter plot by the arrows (gray circles represent individual alternatives). Designs with optimal tradeoffs (Pareto optimal) are indicated by triangles. State-of-the-practice (SOTP) design generated by a COTS ORM system indicated with a red/dark star. Note that the SOTP design is far from optimal, and only a small fraction of designs present optimal tradeoffs. Systematic synthesis and evalution of the entire model space is required to find the optimal tradeoffs.

The above example—which we use as a running example—manifests one of the most prominent and widely-used software/system engineering problems that requires model space synthesis and exploration to be addressed effectively. Model space synthesis is an indispensable part of practical design. The motivation for this paper is the current lack of adequate scientific foundations and practical technologies for scalable model space synthesis in software and systems engineering. The consequences are significant, in opportunity costs, stakeholder dissatisfaction, and in underperforming and failed projects and systems. We hypothesize, and our experimental results confirm, the possibility of a parallelized model space synthesis approach, backed by unsupervised learning, to automatically derive a balanced partitioning of gigantic design spaces. Such a pragmatic synthesis of the entire model space promises revealing designs that greatly outperform those produced by the existing design tools and provides significant performance improvements to both systems designers and their end-users. In the next section, we provide an overview of Parasol, then describe in detail its approach to address these issues and enable the pragmatic and scalable synthesis of large-scale software model spaces.

3 APPROACH

This section overviews our approach—realized in a tool called Parasol[1]—to effectively parallelize systematic synthesis of large model spaces. The driving innovation of this approach is to first synthesize a bounded *sample* of the model space and *cluster* it using unsupervised learning. An invariant from each cluster is then automatically inferred to support a balanced partitioning of the large model space *prior* to exploring the entire set of design variants, allowing each partition to be synthesized concurrently and independently by a distinct synthesis engine. Parasol comprises four main steps, as depicted in Figure 2.

(1) **Sampling**, which automatically synthesizes a bounded subset of models $\mathcal{M}_s$ apropos formula $\mathcal{F}_b$, derived from the system specification, $\mathcal{F}$, via declarative slicing;
(2) **Clustering**, which automatically discovers related subsets within the sample using unsupervised learning;
(3) **Partitioning**, which automatically infers an invariant from each cluster, i, and synthesizes constraints c_i corresponding to that invariant; then
(4) **Parellel synthesis**, where each c_i is conjoined with $\mathcal{F}$ to define a set of independently analyzable partitions of the target model space, which can each be explored concurrently to synthesize the entire model space in parallel.

While our goals are broad, for concrete exposition of our ideas, we use relational logic as an example medium of specification to explain our vision in this paper. Relational logic is shown to be a perfect candidate for software abstraction [26, 27].

3.1 Sampling

The first step in Parasol is to synthesize a small, bounded subset of the desired model space which can act as a representative *sample*. Figure 3 provides an overview of the sampling process. To generate this sample, Parasol accepts as input a formal specification of a system (e.g., in Alloy [26]), which is then translated into a format $\mathcal{F}$, appropriate for consumption by an underlying off-the-shelf *solver* and/or model finder. When the solver finds a model, m, which satisfies $\mathcal{F}$, that model is added to the model space.

[1]https://sites.google.com/view/parallel-exploration/home

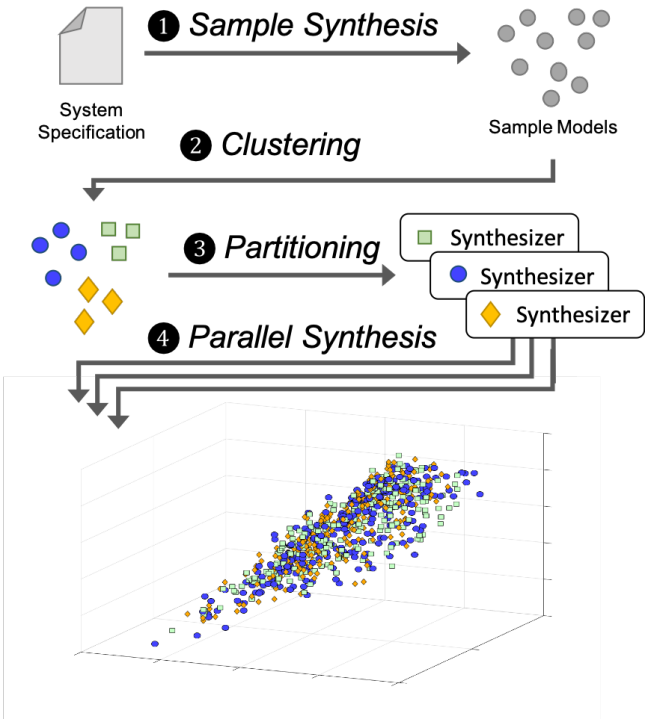


Figure 2: Parasol overview. (1) A *sample* set of models is synthesized from a slice of the specification. (2) The sample set is *clustered* using unsupervised learning. (3) An invariant from each cluster is automatically inferred to support sound *partitioning* of the large model space. (4) Model space *synthesis* is then performed in parallel, greatly reducing the time required to generate the entire model space.

Parasol relies on declarative slicing [55] to identify a base slice for its sampling. Specifically, Parasol first selects a slicing criterion $C \subseteq \mathcal{R}$ and generates a *base slice* and *derived slice* of the specification. The base slice represents a smaller problem than the original due to the removal of all relations that do not appear in the slicing criterion besides all clauses referencing the removed relations. The formulae for the base and derived slices ($\mathcal{F}_b$ and $\mathcal{F}_d$, respectively) partition the formula for the input such that $\mathcal{F} \equiv \mathcal{F}_b \wedge \mathcal{F}_d$. From there, it is easy to deduce that $\mathcal{F} \implies \mathcal{F}_b$; in other words, every assignment of tuples that satisfies $\mathcal{F}$ must also satisfy $\mathcal{F}_b$. This guarantees a mapping between the models of the sample and those of the input specification, as each model of the input is a valid extension of a sample model.

Parasol selects its slicing criterion based on the *complexity* of the formula for the base slice, derived from the number of clauses appearing in the formula. The complexity ratio (Formula 1) compares the number of clauses, C, in the conjunctive normal form (CNF) representation of the original formula with the total number of clauses in the CNF for the base and derived slices (C_b and C_d, respectively). The CNF translation process adds additional auxiliary variables and clauses to represent a given formula as a conjuction of disjunctions. The base and derived slices are each likely to be smaller than the original formula, so their translations will include

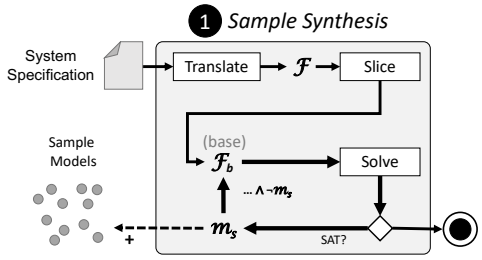


Figure 3: Sample synthesis overview. Input specification is translated into a format suitable for solver, then sliced using declarative slicing. Sample is then synthesized from base slice $\mathcal{F}_b$ using specification-driven model space synthesis.

fewer total auxiliaries than the original formula would require, allowing for complexity ratios greater than one. Parasol filters the possible slicing criteria to those that would produce unique base slices and selects the criterion with the highest complexity ratio.

$$complexity\ ratio = \frac{C}{C_b + C_d} \tag{1}$$

Parasol then uses the specification-driven model space synthesis process to synthesize a sample of model instances satisfying the base slice formula, $\mathcal{F}_b$, as shown in Figure 3. If $\mathcal{F}_b$ is satisfiable, the model finder returns the satisfying model, m_s, and adds it to the sample model space, $\mathcal{M}_s$. A clause representing the *negation* of m_s is conjoined with $\mathcal{F}_b$, and the resulting conjunction is then subjected to another round of analysis to produce a different satisfying solution. This loop continues until the model finder cannot find a satisfying model, completing sample model space synthesis. These sample models satisfy the base slice derived from the original specification. In view of the logical relationship described above between the base slice and the original, larger specification, each model in the original model space, $\mathcal{M}$, is an extension of one of the models in $\mathcal{M}_s$. As such, Parasol can glean from $\mathcal{M}_s$ information about the models of the original specification.

3.2 Clustering

For each model $m_s \in \mathcal{M}_s$, Parasol then generates an *observation vector*, o_s, to provide as input for clustering. This vector is constructed by checking the model against a given set of *features* which serve to identify the relevant similarities and differences

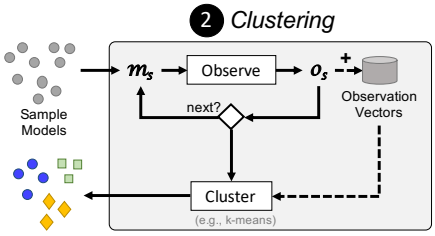


Figure 4: Clustering component overview. For each model m_s in the sample, Parasol constructs an observation vector o_s (see Figure 5). Observations are clustered using unsupervised learning to partition design alternatives in the sample.

$$m_k = \left\{ \begin{array}{ll} r_i = \{t_1^i, t_3^i, \ldots, t_{n-1}^i, t_n^i\} & \longrightarrow [1,0,1,\ldots,1,1] \\ r_j = \{t_2^j, \ldots, t_{m-1}^j, t_m^j\} & \longrightarrow [0,1,\ldots,1,1] \\ \ldots & \ldots \end{array} \right\}_{concat} = o_k$$

$$m_l = \left\{ \begin{array}{ll} r_i = \{t_1^i, t_2^i, \ldots, t_{n-1}^i\} & \longrightarrow [1,1,0,\ldots,1,0] \\ r_j = \{t_1^j, \ldots, t_m^j\} & \longrightarrow [1,0,\ldots,0,1] \\ \ldots & \ldots \end{array} \right\}_{concat} = o_l$$

Figure 5: Example observation vector construction for clustering. Sample models $m_k, m_l \in \mathcal{M}_s$ correspond to observation vectors o_k and o_l, respectively. Each vector is constructed by concatenating sub-vectors corresponding to each relation $r_i, r_j \in \mathcal{R}$, where each index x_α is 1 if tuple $t_x^\alpha \in r_\alpha$ in the corresponding model or 0 otherwise.

among the observations. Parasol uses a list of features extracted from the *tuple assignments* in each satisfying model of the sample problem. Specifically, it defines one Boolean feature for each tuple of atoms t in the *upper bound* of each relation $r \in \mathcal{R}$. The upper bound, denoted r.UpperBound, represents the set of all tuples that *may* be assigned to relation r by the solver in a potential model instance of $\mathcal{F}$, usually defined as the n-fold Cartesian product of the domains of r. The size of each tuple assignment feature vector is therefore equal to $\sum_{r \in \mathcal{R}} | r.\text{UpperBound} |$.

The observation vector for each sample model m_s is computed as shown in Figure 5. First, the relations in $\mathcal{R}$ and the tuples in each relation's upper bound are ordered into a canonical ordering and indexed. The assignments of each relation r_i in m_s is then translated into a sub-vector according to that indexing, setting a value of 1 at each index where the corresponding tuple is a member of r_i in m_s and a value of 0 otherwise. For example, Figure 5 depicts two sample models $m_k, m_l \in \mathcal{M}_s$ and relations $r_i, r_j \in \mathcal{R}$ with cardinalities n and m, respectively. In m_k, $r_i = \{t_1^i, t_3^i, ..., t_{n-1}^i, t_n^i\}$ and $r_j = \{t_2^j, ..., t_{m-1}^j, t_m^j\}$. In this case, the vector o_k generated from m_k would contain a one at indices corresponding to t_1^i and t_{n-1}^i, but would contain a zero at the index corresponding to t_2^i, among others. Similarly, for m_l—which has different tuple assignments for r_i and r_j—the observation vector o_l would contain a one at indices corresponding to t_1^i and t_2^i, but a *zero* at the index for t_3^i.

The observations are then passed through unsupervised clustering algorithms. If the desired degree of parallelism is known, Parasol uses k-means clustering to produce the desired number of clusters. Specifically, if the degree of parallelism is $n > 1$, Parasol will run k-means clustering where $k = min(n-1, 2)$. It generates one fewer cluster in order to account for differences between the sample and the original input, detailed in Section 3.3.

If the desired degree of parallelism is not specified explicitly, Parasol uses x-means clustering [39] to automatically determine an appropriate value for k. X-means clustering determines the best value for k by identifying candidate cluster locations and selecting those that optimize the Bayesian information criterion (BIC) for each cluster. This allows for effective unsupervised clustering of the sample models even when a value for k is not given. The discovered

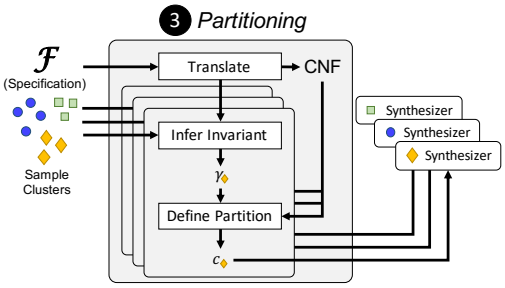


Figure 6: Partitioning component overview. Invariant γ_i is inferred from cluster i, specifying tuples assignments as included/excluded/neither. γ_i is transformed into variable clauses and conjoined to the CNF representation of $\mathcal{F}$ to generate *partition definition* c_i for each synthesis engine.

Algorithm 1 Invariant inference algorithm. Input is a set of relations, $\mathcal{R}$; a set of cluster centroids, each comprising an ordered list of tuple assignment features. Returns invariants for each centroid.

Input: $\mathcal{R}$: relations, *centroids* : set of ordered lists of 0 or 1
Output: *invariants*

```
1:  invariants ← ∅
2:  for c ∈ centroids do
3:      inv ← ⊤
4:      for r ∈ ℛ do
5:          for i = 1 to | r.UpperBound | do
6:              t ← r.UpperBound[i]
7:              if c[i] = 1.0 then
8:                  inv ← inv ∧ (t ∈ r)
9:              else if c[i] = 0.0 then
10:                 inv ← inv ∧ (t ∉ r)
11:     invariants ← invariants ∪ inv
12: return invariants
```

clusters—both the metadata and the list of member observations for each—are provided as input for the next step of our process.

3.3 Partitioning

Parasol logically partitions the *original* model space based on the information derived by the unsupervised learning (shown in Figure 6). It first examines the statistical metadata (e.g., the centroid values) for each cluster to infer a set of *invariants* (i.e., logical statements that are true of every sample model in the cluster) based on the features used for the clustering. Each centroid adopts a value for each feature in the closed range between 0.0 and 1.0, summarizing the feature's values in the sample models assigned to the corresponding cluster. Parasol analyzes each centroid and infers an invariant relational assignment of tuples based on the centroid values. If the value for a particular feature is 1.0 in a given centroid, then Parasol infers that the *assignment* of the tuple corresponding to that feature is invariant in the given cluster. Similarly, a value of 0.0 indicates that *non-assignment* of the tuple is invariant in that cluster.

Alg. 1 outlines the invariant inference process. The algorithm first iterates through a set of centroids representing each cluster (Line 2), generating an invariant for each. The inner loops (Lines 4-5) iterate through each tuple in the upper bound of each relation by

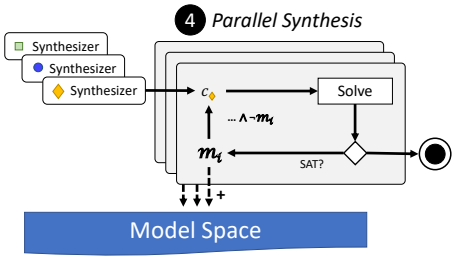


Figure 7: Parallel synthesis component overview. Each synthesizer performs a model space synthesis loop on the provided partition definition. Models satisfying the definition are added to the shared model space, negated, and conjoined with the definition to discover new models. When no more models are found, model space synthesis is complete.

index, using the same index to access to the corresponding feature in centroid c (Lines 7 and 9). If the feature value is equal to 1.0 in the centroid, it conjoins the constraint that the tuple *must* be present in the set defining the relation (i.e. the tuple is *included*); if it is equal to 0.0, it asserts the tuple must *not* be present (i.e., it is *excluded*). For all other values, it adds no clause. Once each the invariant for each cluster have been created, Parasol generates an additional logical definition from the conjunction of the negation of each of the other invariants; this additional definition would match for any models that would not match any of the other invariants, ensuring that Parasol can enumerate all satisfying models.

The resulting invariants logically define each cluster, but would not be directly applicable as constraints for synthesizing models for $\mathcal{F}$; each one explicitly checks for the presence or absence of individual *tuples* inside the previously defined relations, but $\mathcal{F}$ represents the relations themselves as free variables. Therefore, Parasol further translates $\mathcal{F}$ into *conjunctive normal form (CNF)* in order to directly reference variables corresponding to each individual tuple assignment. In the CNF representation, each tuple in the domain of each relation is assigned a *primary variable* indicating that that tuple is assigned to that relation in a given model. By finding the primary variable associated with each relation and tuple referenced in the invariant, Parasol can explicitly add clauses to the CNF requiring the variables for included tuples to be "true" and excluded tuples to be "false". If the invariant does not imply inclusion or exclusion for a given relation and tuple, then no constraint is added for the corresponding primary variable in the CNF. Once every tuple assignment has been checked, the emitted clauses are conjoined with the original CNF to generate a *partition definition* which can be given to an independent synthesis engine.

3.4 Parallel Synthesis

The last step in our approach (cf. ❹ in Figure 2) is to distribute each partition definition—c_i for some partition i—to a separate synthesis engine and synthesize the model space for each partition concurrently. Parasol again uses a variant of the model space synthesis process described in Section 3.1, as shown in Figure 7. If a satisfying model m_i is found for c_i, the model is added to the model space, c_i is extended by conjoining it with the negation of m_i, and the result is given back to the solver/model finder. This process loops until no satisfying model instance can be found, at which point the synthesis terminates. When the analysis of all the partitions has terminated, Parasol will have synthesized the same model space as that synthesized by state-of-the-art model space synthesis approaches, but in a fraction of the time.

4 EVALUATION

This section presents the experimental evaluation of Parasol. Our evaluation addresses the following research questions:

RQ1. How well does Parasol's learning-based, parallelized model space synthesis perform vs state-of-the-art techniques?
RQ2. Does our learning-based partitioning divide the work more evenly than competing model space partitioning methods?
RQ3. How much overhead is introduced by Parasol?

Experimental setup. We conducted the experiments using a custom Java 13 implementation of Parasol[2], comprising over 5,000 lines of code using the built-in parallel streaming capabilities of Java 8 and above [38] to execute concurrently on multiple threads. The learning-based partitioning algorithms are realized on top of the WEKA library [20] developed by the Machine Learning Group at the University of Waikato. The specifications used in the experiments were developed in Alloy relational logic specification language [27] and executed using the Java API of Alloy 5, the Kodkod model finder [54] which drives that version of the Alloy Analyzer, and the Glucose 4.1 SAT solver [2–4]. All experiments were run on OpenStack instances running Ubuntu 20.04, each with 16 VCPUs and 60GB RAM.

Subject systems. We collected a set of eight system specifications as our experimental subjects, representing model space exploration problems in three different domains: (1) *database schema design*, and more specifically as explained in Section 2, object-relational database mapping (ORM) design; (2) *role engineering* within a role-based access control (RBAC) system; and (3) *security analysis*, where the threat space of real-world IoT apps, scoped by threat models thereof, should be exhaustively explored in order to help discover and address the risks. All subjects—as well as our reference implementation of Parasol and experimental data—are publicly available online for reuse [5].

Database Design. Our first set of subjects evaluates Parasol in the context of designing *object-relational database mapping (ORM)* schemas. The ORM tools provide an indirection layer between object-oriented data models and relational database management systems, and are included in many popular libraries (e.g., Hibernate [18]) and frameworks (e.g., Django [16]). The design mapping strategies employed by an ORM can have a large impact on its performance. Selecting an appropriate design often demands the analysis of tradeoffs among candidates, requiring analysis of the entire model space. We considered the database design problem of two systems, adopted from the literature. The first is the object model of an E-commerce system adopted from Lau and Czarnecki [30]. It represents a common architecture for open-source and commercial E-commerce systems. The other object model is for a cyber-social operating system, CSOS [6], to help coordinate people and tasks.

[2]Research artifacts and experimental data are available at https://sites.google.com/view/parallel-exploration/home

Table 1: Experimental data for each subject system specification. Average total runtime (limited to 24 hours) for three executions reported for (a) model space synthesis performed using the approach from TradeMaker [6, 7] with state-of-the-art incremental and parallel SAT solvers and (b) using Parasol. Model space size is the number of models synthesized by the TradeMaker baseline for the full system specification. Runtime column for Parasol includes overhead.

Specification Domain	System	Model Space Size	Model Space Synthesis [6, 7]		Parasol		
			Incremental Solver [2]	Parallel Solver [11]			
			Runtime (secs)	Runtime (secs)	Runtime (secs)	Overhead (secs)	Speedup
Database Design	E-commerce [30]	803,863	36,968.49	>86,400.00	14,179.25	633.89	250%
	CSOS [6]	1,576,796	81,142.15	>86,400.00	17,020.02	349.14	467%
Role Eng.	RBAC [46]	362,133	1,768.02	>86,400.00	233.38	0.71	755%
Security	IoT Threats [1] (App Bundles 1–5)	130,816	2,888.78	>86,400.00	712.95	86.41	361%
		301,360	16,389.06	>86,400.00	6,183.67	695.23	276%
		999,424	42,041.08	>86,400.00	6,223.16	699.46	607%
		958,464	39,568.51	>86,400.00	5,951.50	761.37	589%
		1,042,688	53,511.07	>86,400.00	8,529.01	602.78	586%

Role Engineering. The next subject is centered around model space synthesis and analysis of the tradeoffs in developing roles and their associated permissions within a *role-based access control (RBAC)* system [46], a process known as *role engineering*. In such a system, a given user has a set of permissions and resources that are required to perform their job as well as a set of roles that have been assigned to that user. In each satisfying assignment, each user must be assigned roles that grant their required permissions/resources. System administrators must then select the roles that provide the best tradeoffs among various non-functional properties, such as minimizing the number of roles they will need administer or the number of unneeded permissions assigned to users in the system. The analysis allows administrators to select the role assignment that best satisfies the desired qualities.

Security Analysis. Our last set of subjects represents discovering and exploring the space of potential security threats in IoT systems. Each warning produced denotes a possible threat to the security of the IoT system. Fully assessing the risk and mitigating the impact of each threat would require examining every violations. Ensuring the security of the system therefore requires synthesis of the entire threat space to discover and address all possible security risks. To evaluate Parasol's performance in this context, we considered 100 real-world IoT app models drawn from [1]. We divided the apps collection into five non-overlapping groups of 20 each, and explored for bundles of apps among each group that violate interaction threat assertions. Each individual model in the model space corresponds to a single possible threat that may arise from interactions among apps within the corresponding bundle.

Baselines and Measures. To evaluate Parasol in the context of the research questions, we synthesized model spaces (i.e., design models, role models, and threat models) for each subject system and measured (a) the total execution runtime to synthesize the entire model space, in seconds; (b) the overhead incurred by Parasol, in seconds; and (c) the number of models synthesized by each analysis engine during execution, used to compute the *coefficient of variation* to measure partition parity.

To answer RQ1, we synthesized the model space for each subject specification with Parasol and compared the results against synthesis using the model space synthesis method described in TradeMaker [6, 7], the state-of-the-art in systematic, specification-driven model space synthesis. Our baseline approach relies upon SAT solvers to explore the space of models for various systems specifications provided as input. To ensure a fair comparison against TradeMaker, we empowered it with state-of-the-art SAT solvers. Specifically, we studied two classes of SAT solvers as the underlying analysis engine for the baseline approach. First, we used an *incremental* SAT solver to directly represent the TradeMaker approach as described in [6, 7]. Second, we used a *parallel* solver, which allows us to compare the high-level parallelization approach taken by Parasol to lower-level, CNF-based parallelization.

To evaluate RQ2, we synthesized the model space for each subject in parallel using two other partitioning methods as our baselines, described in more detail in Section 4.2. Lastly, for RQ3 we measured the overhead for synthesis with Parasol against the overhead for the state-of-the-art model space synthesis. In order to ensure our experiments completed within a reasonable time, we limited each execution—both for Parasol and each baseline—to 24 hours. The model space for each system was synthesized three times with Parasol and with each baseline, and the mean values are reported. The resulting model spaces synthesized by each approach are all the same, as Parasol synthesizes the same models as state-of-the-art approaches in a much shorter time.

4.1 RQ1: Parasol In Practice

To answer the first research question, we compared the time taken to synthesize the model space of each experimental subject with Parasol to that required to synthesize the same model spaces using TradeMaker, the state-of-the-art model space synthesis technique [6, 7]. We used two state-of-the-art SAT solvers to drive our baseline: (a) Glucose [3]—an *incremental* SAT solver; and (b) Plingeling [11]—a *parallel* SAT solver. Using the incremental solver provides a direct comparison with our baseline, as an incremental

SAT solver is used by TradeMaker as presented in [6, 7]. The parallel solver allows us to compare our approach against lower-level parallelization performed by the SAT solver. We used our reference Java 13 implementation of Parasol, and collected the running time using the Bash `time` command. To ensure a fair evaluation, we used Plingeling's default degree of parallelism (8) as the number of partitions in the experiments.

The experimental results are summarized in Table 1. Across the board, Parasol outperformed the incremental baseline, providing an average speedup of 460%. The largest speedup (755%) was obtained on the role engineering model, which also had the shortest overall running time.

The IoT threat space enumeration subjects fell in the middle, with an average speedup of 483%; the speedup was more pronounced on app bundles with more potential security risks. For example, Parasol shows 586% speedup in the analysis of Bundle 5 with over a gigantic number of potential threats detected vs. Bundle 1 with only 130,816 potential threats and 361% speedup. Lastly, Parasol demonstrates the average speedup of 358% for the two ORM database design problems. As a case in point, for CSOS—the system with the most valid database design models—Parasol saved the most total hours compared to the baseline, reducing runtime from 22.5 hours to under 5 hours.

Parasol also performed well against the baseline approach driven by the Plingeling parallel SAT solver. As shown in Table 1, the baseline using the parallel solver was unable to fully synthesize *any* of the model spaces within a 24-hour time period. This is in part due to the fact that the underlying parallel solvers are intended to quickly provide a *single* SAT/UNSAT determination by dividing a given CNF problem into subproblems and executing in parallel. Model space synthesis techniques such as that used by our baseline, however, repeatedly invoke the underlying solver, incrementing the previous formula with a new clause for each discovered design variant. Plingeling treats each such new invocation as an entirely new problem, discarding any information discovered during the solve steps from previous iterations. In contrast, Parasol extracts specification-level domain knowledge from each model to partition the problem at a higher level of abstraction. Because the parallel-SAT version of the state-of-the-art baseline approach was unable to synthesize any of the subject model spaces within 24 hours, we excluded that version from subsequent experiments and used only the incremental-SAT.

> Overall, the experimental results indicate that Parasol provides significant time savings—with an overall average speedup of 460%—compared to state-of-the-art model space synthesis techniques using an incremental SAT solver.

4.2 RQ2: Parasol vs. Other Partitioners

To address the second research question, we set out to assess how well Parasol's learning-based partitioning divides the model space and whether it distributes the work more evenly than competing specification-level partitioning methods. We considered two competing partitioning approaches, comparing each against our learning-based partitioning (c.f. Section 3):

(1) **Random partitioning:** With *random* partitioning, each free variable in the original specification is—with equal probability—explicitly included, excluded, or neither, in which case that variable is left to the solver to include or exclude. To test this method of partitioning, we implemented an algorithm where the the random partitioner elects whether to include, exclude, or defer each variable based on a value drawn from a uniform random distribution rather than setting the variables in the CNF according to the invariants.

(2) **Scope partitioning:** *Scope* partitioning relies on information already encoded in the input specification to guide the partitioning. Formal specifications for bounded verification tools, like Alloy, include constraints on the *scope* of the analysis. Specifically, the author of the specification determines the scope by declaring the maximum number of distinct atoms that can be assigned to certain type-like unary relations (i.e., *sigs*). For example, in the ORM design specifications, the user determines the maximum number of tables that may appear in each model by setting the scope for the Table *sig*. For the CSOS specification, the Table scope is set to 18, meaning that the solver will explore models containing between 0 and 18 tables. For the scope partitioning baseline, we restrict each partition to synthesize models within a subrange of the original scopes; for example, one partition might synthesize all models of CSOS with exactly 18 tables, another all models with exactly 17 tables, and so on. We implemented an iterative partitioner that loops through each type-like relation in the specification, creating partitions by fixing the scope of each relation to a single value until the desired number of partitions has been created.

The boxplots in Figure 8 show the synthesis time (in seconds) taken by each of the techniques vs. Parasol over the subject systems. The horizontal axis specifies the synthesis methods: TradeMaker's model space synthesis with an *incremental* solver (Glucose [3]), parallel synthesis using *random* partitioning, parallel synthesis using *scope* partitioning, and Parasol (orange, right-most box). Parasol tends to exhibit significantly lower synthesis time compared to the other techniques.

To determine how each of the three partitioning methods divides the model space among the parallel analysis engines, we tracked the number of models synthesized by each distributed worker using each of the three partitioning methods. We then computed the mean (μ) and standard deviation (σ) of the number of models synthesized for each partition for each method in order to calculate the coefficient of variation (C_v)—the ratio of standard deviation to mean—as a measure of the parity among the partitions. The C_v assumes a value between zero and the square root of the number of partitions (i.e., 8), with a lower C_v indicating more balanced division of work.

Table 2 presents the results for each partitioning method and subject specification. Parasol significantly outperformed both the random and scope-based partitioning baselines in all but one case, where the partitions produced by scope-based partitioning are the same as those produced by Parasol; for the role engineering subject the slicing criterion used by Parasol contained only the Role *sig* from the specification, resulting in the same partitions as those created by the scope partitioning baseline. In terms of partition parity, the C_v for Parasol was less than half of the square root of the number of partitions for each subject system, indicating that Parasol evenly divides work among the partitions. Furthermore,

Table 2: Total runtime (in seconds, including overhead), speedup, and coefficient of variation (C_v) across partitions for each subject system specification from each of the three partitioning methods. Note that 2.83 is the maximum value for C_v, and results when one worker synthesizes the entire model space. Parasol's learning-based partitioning produces significantly more balanced partitions (lower C_v) and substantially less runtime than other partitioning methods for all subjects.

Partitioning Method	Metric	Subject System							
		E-commerce	CSOS	RBAC	IoT Threats				
Random	Runtime (secs)	38,865.65	84,675.95	1,794.32	2,820.19	14,818.77	40,728.11	37,127.55	54,506.31
	C_v	2.83	2.83	2.83	2.83	2.83	2.83	2.83	2.83
Scope	Runtime (secs)	36,447.39	86,744.50	**229.82**	2,957.37	15,590.44	40,440.00	39,126.65	50,559.78
	C_v	2.83	2.83	**1.35**	2.83	2.83	2.83	2.83	2.83
Parasol	Runtime (secs)	**14,813.14**	**17,369.16**	234.09	**799.36**	**6,878.90**	**6,922.62**	**6,712.87**	**9,131.79**
	C_v	**0.87**	**1.48**	**1.35**	**0.50**	**0.67**	**0.57**	**0.54**	**0.52**
Speedup w/ Parasol	*vs. Random*	262%	488%	767%	353%	215%	588%	553%	597%
	vs. Scope	245%	499%	98%	370%	227%	584%	582%	554%

Parasol had the lowest C_v among all three partitioning methods for all specifications, showing that Parasol outperforms the baseline approaches in terms of evenly partitioning the model space.

We interpret these data to suggest that Parasol improves upon other partitioning methods for (a) runtime performance and (b) even partitioning of work among parallel workers.

4.3 RQ3: Overhead

To determine the overhead incurred by Parasol, we computed the time (in seconds) between the start of execution and the start of the exploration of the model space for the original specification for both the state-of-the-art model space synthesis used as a baseline for RQ1 and Parasol. That overhead time period includes the sampling and partitioning conducted by Parasol before design space exploration, which is not performed by state-of-the-art model space synthesis. The overhead, then, can be represented as the difference between those two time durations. Table 1 summarizes the overhead (in seconds) for each of subject in the last column. Overall, the execution time overhead incurred by Parasol accounted for a small fraction of the running time (< 7% on average), making the effect on user experience negligible.

Across the board, the speedup provided by Parasol substantially outweighs the overhead.

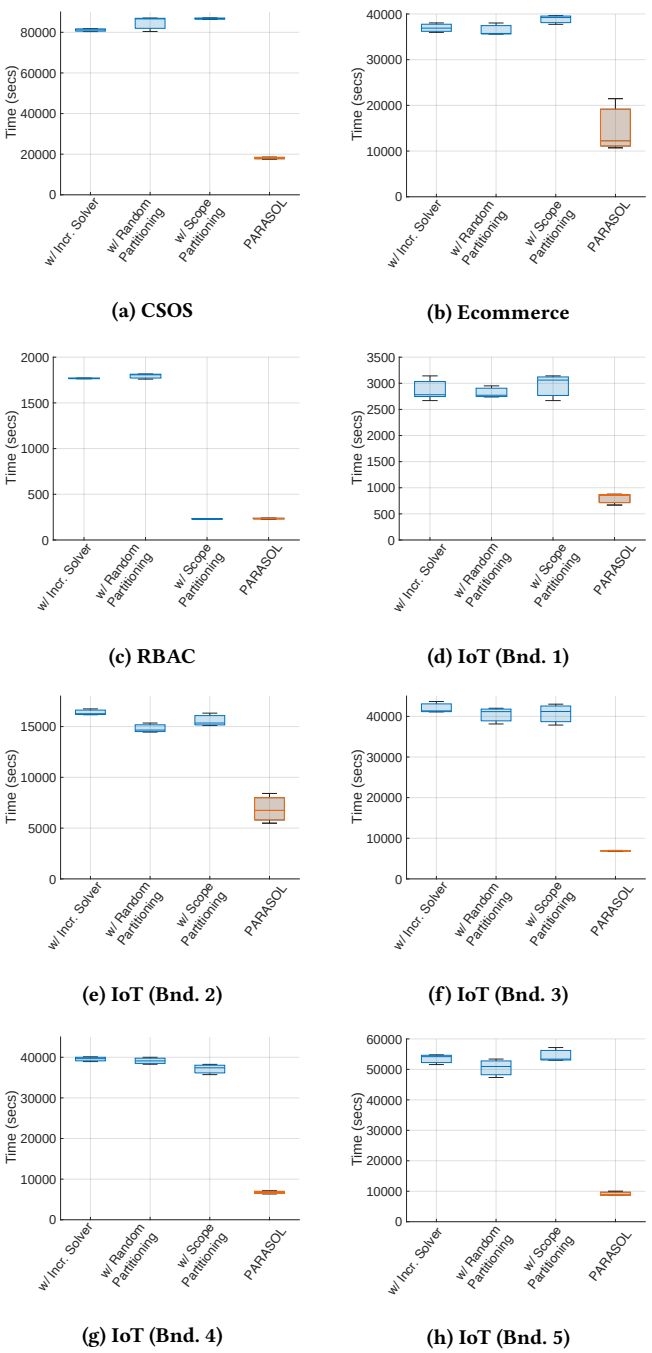
5 DISCUSSION

Overall, the results described in Section 4 demonstrate that Parasol can synthesize a huge model space much more efficiently than state-of-the-art model space synthesis techniques, exhibiting an average speedup of over 460% compared to the competing approaches. Parasol also outperforms state-of-the-art parallel solvers by partitioning the model space at a higher level of abstraction than the underlying SAT problem. We interpret our results to show that the variance among partitions produced by Parasol is low overall, indicating the effectiveness of our learning-based partitioning in evenly dividing work among the parallel workers (see Table 2). Finally, the overhead required by Parasol is minimal considering the multiplicative speedup Parasol provides compared to state-of-the-art model space synthesis (see Table 1).

For some experimental subjects (e.g., CSOS), the variation among the partitions was higher than the others. We believe this was due to differences in how well the sample represented the implicit structure of the specification. Parasol samples via declarative slicing using the criteria described in Section 3, which measures the complexity ratio of the base slice compared to the original formula (see Formula 1). For some specifications, the base slice effectively matched the high-level structure of the target model space. For example, the base slice selected for all five IoT Coordination Threat app groups included the relation defining which apps were installed. Each partition was thus defined by the inclusion/exclusion of specific combinations of apps rather than by potential security risks, resulting in a low coefficient of variation (between 0.50 and 0.67).

In contrast, the slice selected for the RBAC specification partitioned based on the number of roles included in the models (i.e., one partition synthesized all models with exactly one role, another all variants containing exactly two roles, etc.). This resulted in greater imbalance among the partitions. For example, there are more possible assignments of roles and permissions in a system with five roles than in a system with only two. This is indicated by the higher C_v (1.35) for the RBAC system. It is also worth noting that the slice selected by Parasol for that particular system resulted in exactly the same partitioning as the scope-based partitioning method, hence the two methods have the same C_v. This was the only case among all eight subject specifications where the scope-based partitioning provided the same partitions. In all other cases, Parasol outperformed the other methods both in terms of runtime performance and allocation of work among partitions.

The formulae selected during declarative slicing also determined the number of models synthesized for the sample, which was the largest contributor to the overhead for each subject specification.



(a) CSOS (b) Ecommerce

(c) RBAC (d) IoT (Bnd. 1)

(e) IoT (Bnd. 2) (f) IoT (Bnd. 3)

(g) IoT (Bnd. 4) (h) IoT (Bnd. 5)

Figure 8: Boxplots depicting the synthesis time (in seconds) taken by each of the techniques vs. Parasol over the subject systems. Horizontal axis indicates synthesis method: state-of-the-art model space synthesis with an *incremental* solver (Glucose [3]), parallel synthesis using *random* partitioning, parallel synthesis using *scope* partitioning, and Parasol (orange, right-most box).

For example, the "installed app" slice chosen for the IoT Coordination Threat app groups generated more than 300,000 sample instances, increasing the overhead. On the other hand, the base slices generated for RBAC produced only the scope of the *Role* signature in the specification, despite the large number of models generated for the problem overall. Further research into different slicing algorithms could improve Parasol's ability to select smaller and/or more representative samples, leading to even greater speedups.

6 THREATS TO VALIDITY

The main threat to the internal validity of our experimental evaluation is the accuracy of our custom implementation. Our code was debugged and tested thoroughly, including unit tests written with JUnit to ensure our implementation synthesized models correctly. We also canonicalized and serialized the models synthesized by Parasol and each baseline to ensure the same model space was synthesized by each one; the de-duplicated set of models was the same in each case, providing evidence that our implementation works as intended. To ensure our external validity, we have conducted our experiments on real-world model spaces drawn from various software engineering domains. Lastly, the validity of our construct relies on the use of the coefficient of variation as a proxy for parity among our partitions; there may be other possible measures, such as standard error of the mean or median absolute deviation. C_v is used in a wide variety of statistical and analytical settings, and provides an intuitive understanding of the variation, so we believe it is a reasonable metric to use for our analysis.

7 RELATED WORK

Researchers have explored the area of specification-driven design space exploration. Kang, Jackson, and Schulte [28] employed formal methods for design space exploration using an SMT solver, but focused the effort on finding one or more satisfying model—not necessarily the entire model space. The focus of more recent research efforts is usually on constraining the model space in order to filter some models during a complete exploration. Sullivan et al. [51] recently proposed a general abstraction idiom to only synthesize models that satisfy a given abstraction function. Porncharoenwase et al. [40] presented an approach that finds a representative sample, attempting to demonstrate the most syntactic coverage while exploring the smallest fraction of the overall space. Nelson et al. [36] developed an approach called Aluminum that enables users to manually guide exploration, which may not be suitable for large-scale systematic analysis. These approaches all seek to limit the model space rather than to effectively explore all variants, assuming that the user performing the analysis can determine which variants are irrelevant. Parasol makes no such assumption, providing significant speedups while still exploring the entire model space.

Rosner et al. developed Ranger [42] to partition a bounded model checking problem based on ranges of bound assignments. However, Ranger focuses on finding *one* instance or counterexample rather than parallelizing the model space exploration. In fact, the recursive range partitioning technique used by Ranger to ensure parity among the worker nodes—which introduces acceptable overhead for finding a single model—would be invoked too frequently to be suitable for exploring vast model spaces. Portfolio solvers, such as

ManySAT [23] and HordeSAT [9], attempt many different configurations of sequential solvers in parallel to find the best method of solving the problem. Iser et al. [25] described an approach that can aid such solvers by using a shared repository of clauses to mitigate memory issues. As demonstrated in Section 4, the techniques used in these parallel (but non-incremental) solvers (e.g., Plingling [11]) optimize finding a *single* satisfying model, leading to extremely poor performance when searching for subsequent models. *Parasol avoids these pitfalls by using learning at a higher level to automatically derive domain knowledge, guiding the parallelization of exploring the model space.*

Lastly, while other constraint solvers (e.g., SMT solvers like Z3 [15]) have made impressive improvements in recent years, they still lag behind SAT-based solutions when analyzing relational specifications. Meng et al. [32] developed an approach to use CVC4 [10] as the underlying solver for Alloy specifications, but the SAT-based solution still outperformed their implementation for most specifications. Stoel et al. [50] developed a Z3-based tool which was similarly less efficient than the baseline Alloy translation to SAT. Therefore, we implement Parasol atop a SAT solver, as that represents the current state-of-the-art for relational specifications.

8 CONCLUSION

In this paper, we presented Parasol, a novel approach to perform systematic specification-driven model space synthesis in parallel, leveraging unsupervised learning to extract domain knowledge about specification in order to evenly partition the model space. The experimental results demonstrated an average speedup of 460% over state-of-the-art model space synthesis, possibly saving hours or even days for large system specifications. The results further corroborated that the division of work guided by our learning-based partitioning strategy was more even than competing partitioning strategies. We also showed that the fractional overhead introduced by the sampling is far outweighed by the improvement in overall runtime.

In future research, we would seek to explore some of the components of the approach to improve upon this research. For example, different strategies may be employed to extract samples with declarative slicing, rather than using the complexity ratio. Furthermore, parallel synthesis of the model space could also enable further optimizations, such as conducting dynamic tradespace analysis where each of the design variants is evaluated in parallel as well. Finally, we also believe further research could be done using different learning techniques—perhaps supervised learning with some form of oracle—to better extract domain knowledge.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable comments. This work was completed utilizing the Holland Computing Center of the University of Nebraska, which receives support from the Nebraska Research Initiative. This work was supported in part by awards CCF-1755890, CCF-1618132, CCF-2139845, and CCF-2124116 from the National Science Foundation.

REFERENCES

[1] Mohannad Alhanahnah, Clay Stevens, and Hamid Bagheri. 2020. Scalable analysis of interaction threats in IoT systems. In *ISSTA '20: 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, USA, July 18-22, 2020*, Sarfraz Khurshid and Corina S. Pasareanu (Eds.). ACM, 272–285. https://doi.org/10.1145/3395363.3397347

[2] Gilles Audemard, Jean-Marie Lagniez, and Laurent Simon. 2013. Improving Glucose for Incremental SAT Solving with Assumptions: Application to MUS Extraction. In *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings (Lecture Notes in Computer Science)*, Matti Järvisalo and Allen Van Gelder (Eds.), Vol. 7962. Springer, 309–317. https://doi.org/10.1007/978-3-642-39071-5_23

[3] Gilles Audemard and Laurent Simon. 2009. Predicting Learnt Clauses Quality in Modern SAT Solvers. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, Craig Boutilier (Ed.). 399–404. http://ijcai.org/Proceedings/09/Papers/074.pdf

[4] Gilles Audemard and Laurent Simon. 2018. On the Glucose SAT Solver. *Int. J. Artif. Intell. Tools* 27, 1 (2018), 1840001:1–1840001:25. https://doi.org/10.1142/S0218213018400018

[5] Authors. 2020. Project website. https://sites.google.com/view/parallel-exploration/home.

[6] Hamid Bagheri, Chong Tang, and Kevin J. Sullivan. 2014. TradeMaker: automated dynamic analysis of synthesized tradespaces. In *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*. 106–116. https://doi.org/10.1145/2568225.2568291

[7] Hamid Bagheri, Chong Tang, and Kevin J. Sullivan. 2017. Automated Synthesis and Dynamic Analysis of Tradeoff Spaces for Object-Relational Mapping. *IEEE Trans. Software Eng.* 43, 2 (2017), 145–163. https://doi.org/10.1109/TSE.2016.2587646

[8] Georgios Bakirtzis, Brandon J. Simon, Aidan G. Collins, Cody Harrison Fleming, and Carl R. Elks. 2020. Data-Driven Vulnerability Exploration for Design Phase System Analysis. *IEEE Syst. J.* 14, 4 (2020), 4864–4873. https://doi.org/10.1109/JSYST.2019.2940145

[9] Tomás Balyo, Peter Sanders, and Carsten Sinz. 2015. HordeSat: A Massively Parallel Portfolio SAT Solver. *CoRR* abs/1505.03340 (2015). arXiv:1505.03340 http://arxiv.org/abs/1505.03340

[10] Clark W. Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. 2011. CVC4. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings (Lecture Notes in Computer Science)*, Ganesh Gopalakrishnan and Shaz Qadeer (Eds.), Vol. 6806. Springer, 171–177. https://doi.org/10.1007/978-3-642-22110-1_14

[11] Armin Biere. 2017. CaDiCaL, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2017. In *Proc. of SAT Competition 2017 – Solver and Benchmark Descriptions (Department of Computer Science Series of Publications B)*, Tomáš Balyo, Marijn Heule, and Matti Järvisalo (Eds.), Vol. B-2017-1. University of Helsinki, 14–15.

[12] Javier Cámara, David Garlan, and Bradley R. Schmerl. 2017. Synthesis and Quantitative Verification of Tradeoff Spaces for Families of Software Systems. In *Proceedings of ECSA*.

[13] Paul D. Collopy. 2018. Tradespace Exploration: Promise and Limits. In *Disciplinary Convergence in Systems Engineering Research*, Azad M. Madni, Barry Boehm, Roger G. Ghanem, Daniel Erwin, and Marilee J. Wheaton (Eds.). Springer International Publishing, 297–307.

[14] Byron Cook. 2018. Formal Reasoning About the Security of Amazon Web Services. In *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I (Lecture Notes in Computer Science)*, Hana Chockler and Georg Weissenbacher (Eds.), Vol. 10981. Springer, 38–47. https://doi.org/10.1007/978-3-319-96145-3_3

[15] Leonardo Mendonça de Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings (Lecture Notes in Computer Science)*, C. R. Ramakrishnan and Jakob Rehof (Eds.), Vol. 4963. Springer, 337–340. https://doi.org/10.1007/978-3-540-78800-3_24

[16] Django Foundation. 2020. Django website. https://www.djangoproject.com/.

[17] Tobias Dürschmid, Eunsuk Kang, and David Garlan. 2019. Trade-off-oriented development: making quality attribute trade-offs first-class. In *Proceedings of ICSE-NIER*.

[18] Steve Ebersole, Gail Badner, Andrea Boriero, and Sanne Grinovero. 2020. Hibernate website. https://hibernate.org/orm/.

[19] Eduardo B. Fernández. 2016. Threat Modeling in Cyber-Physical Systems. In *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress,*

DASC/PiCom/DataCom/CyberSciTech 2016, Auckland, New Zealand, August 8-12, 2016. IEEE Computer Society, 448–453. https://doi.org/10.1109/DASC-PICom-DataCom-CyberSciTec.2016.89

[20] Eibe Frank, Mark A. Hall, Geoffrey Holmes, Richard Kirkby, Bernhard Pfahringer, Ian H. Witten, and Len Trigg. 2010. Weka-A Machine Learning Workbench for Data Mining. In *Data Mining and Knowledge Discovery Handbook, 2nd ed*, Oded Maimon and Lior Rokach (Eds.). Springer, 1269–1277. https://doi.org/10.1007/978-0-387-09823-4_66

[21] Richard P. Gabriel. 2006. Design beyond Human Abilities. In *Proceedings of the 5th International Conference on Aspect-Oriented Software Development (AOSD '06)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/1119655.1119658

[22] Juan P. Galeotti, Nicolás Rosner, Carlos Gustavo López Pombo, and Marcelo F. Frias. 2013. TACO: Efficient SAT-Based Bounded Verification Using Symmetry Breaking and Tight Bounds. *IEEE Trans. Software Eng.* 39, 9 (2013), 1283–1307. https://doi.org/10.1109/TSE.2013.15

[23] Youssef Hamadi, Saïd Jabbour, and Lakhdar Sais. 2009. ManySAT: a Parallel SAT Solver. *J. Satisf. Boolean Model. Comput.* 6, 4 (2009), 245–262. https://satassociation.org/jsat/index.php/jsat/article/view/77

[24] David Heinemeier Hansson. 2020. Ruby on Rails website. https://rubyonrails.org/.

[25] Markus Iser, Tomás Balyo, and Carsten Sinz. 2019. Memory Efficient Parallel SAT Solving with Inprocessing. In *31st IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2019, Portland, OR, USA, November 4-6, 2019*. IEEE, 64–70. https://doi.org/10.1109/ICTAI.2019.00018

[26] Daniel Jackson. 2002. Alloy: A Lightweight Object Modelling Notation. *ACM Trans. Softw. Eng. Methodol.* 11, 2 (April 2002), 256–290. https://doi.org/10.1145/505145.505149

[27] D. Jackson. 2012. *Software Abstractions* (2nd ed.). MIT Press.

[28] Eunsuk Kang, Ethan K. Jackson, and Wolfram Schulte. 2010. An Approach for Effective Design Space Exploration. In *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems - 16th Monterey Workshop 2010, Redmond, WA, USA, March 31- April 2, 2010, Revised Selected Papers (Lecture Notes in Computer Science)*, Radu Calinescu and Ethan K. Jackson (Eds.), Vol. 6662. Springer, 33–54. https://doi.org/10.1007/978-3-642-21292-5_3

[29] Sun Kim and Hantao Zhang. 1994. ModGen: Theorem Proving by Model Generation. In *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1*, Barbara Hayes-Roth and Richard E. Korf (Eds.). AAAI Press / The MIT Press, 162–167. http://www.aaai.org/Library/AAAI/1994/aaai94-025.php

[30] Sean Quan Lau. 2006. *Domain Analysis of E-Commerce Systems Using Feature-Based Model Templates*. Master's thesis. University of Waterloo.

[31] Dong Liu and Benjamin Carrión Schäfer. 2016. Efficient and reliable High-Level Synthesis Design Space Explorer for FPGAs. In *Proceedings of FPL*. 1–8.

[32] Baoluo Meng, Andrew Reynolds, Cesare Tinelli, and Clark W. Barrett. 2017. Relational Constraint Solving in SMT. In *Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings (Lecture Notes in Computer Science)*, Leonardo de Moura (Ed.), Vol. 10395. Springer, 148–165. https://doi.org/10.1007/978-3-319-63046-5_10

[33] Sanjai Narain. 2013. ConfigAssure: A Science of Configuration. In *32th IEEE Military Communications Conference, MILCOM 2013, San Diego, CA, USA, November 18-20, 2013*, Joe Senftle, Mike Beltrani, and Kari Karwedsky (Eds.). IEEE, 1497–1498. https://doi.org/10.1109/MILCOM.2013.252

[34] Luigi Nardi, David Koeplinger, and Kunle Olukotun. 2019. Practical Design Space Exploration. In *27th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS 2019, Rennes, France, October 21-25, 2019*. IEEE Computer Society, 347–358. https://doi.org/10.1109/MASCOTS.2019.00045

[35] Timothy Nelson, Christopher Barratt, Daniel J. Dougherty, Kathi Fisler, and Shriram Krishnamurthi. 2010. The Margrave Tool for Firewall Analysis. In *Uncovering the Secrets of System Administration: Proceedings of the 24th Large Installation System Administration Conference, LISA 2010, San Jose, CA, USA, November 7-12, 2010*, Rudi van Drunen (Ed.). USENIX Association. https://www.usenix.org/conference/lisa10/margrave-tool-firewall-analysis

[36] Tim Nelson, Salman Saghafi, Daniel J. Dougherty, Kathi Fisler, and Shriram Krishnamurthi. 2013. Aluminum: principled scenario exploration through minimality. In *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, David Notkin, Betty H. C. Cheng, and Klaus Pohl (Eds.). IEEE Computer Society, 232–241. https://doi.org/10.1109/ICSE.2013.6606569

[37] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. 2015. How Amazon Web Services Uses Formal Methods. *Commun. ACM* 58, 4 (mar 2015), 66–73. https://doi.org/10.1145/2699417

[38] Oracle. 2019. Java Tutorials – Parallelism. https://docs.oracle.com/javase/tutorial/collections/streams/parallelism.html.

[39] Dan Pelleg and Andrew W. Moore. 2000. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, Pat Langley (Ed.). Morgan Kaufmann, 727–734.

[40] Sorawee Porncharoenwase, Tim Nelson, and Shriram Krishnamurthi. 2018. CompoSAT: Specification-Guided Coverage for Model Finding. In *Formal Methods - 22nd International Symposium, FM 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 15-17, 2018, Proceedings (Lecture Notes in Computer Science)*, Klaus Havelund, Jan Peleska, Bill Roscoe, and Erik P. de Vink (Eds.), Vol. 10951. Springer, 568–587. https://doi.org/10.1007/978-3-319-95582-7_34

[41] Alastair Reid, Luke Church, Shaked Flur, Sarah de Haas, Maritza Johnson, and Ben Laurie. 2020. Towards making formal methods normal: meeting developers where they are. *CoRR* abs/2010.16345 (2020). arXiv:2010.16345 https://arxiv.org/abs/2010.16345

[42] Nicolás Rosner, Junaid Haroon Siddiqui, Nazareno Aguirre, Sarfraz Khurshid, and Marcelo F. Frias. 2013. Ranger: Parallel analysis of alloy models by range partitioning. In *Proceedings of ASE*.

[43] Adam Ross, Hugh McManus, Donna Rhodes, and Daniel Hastings. [n.d.]. *Revisiting the Tradespace Exploration Paradigm: Structuring the Exploration Process*. https://doi.org/10.2514/6.2010-8690

[44] Adam M. Ross, Daniel E. Hastings, Joyce M. Warmkessel, and Nathan P. Diller. 2004. Multi-Attribute Tradespace Exploration as Front End for Effective Space System Design. *Journal of Spacecraft and Rockets* 41, 1 (2004), 20–28. https://doi.org/10.2514/1.9204

[45] Kathrin Rosvall and Ingo Sander. 2018. Flexible and Tradeoff-Aware Constraint-Based Design Space Exploration for Streaming Applications on Heterogeneous Platforms. *ACM Trans. Design Autom. Electr. Syst.* 23, 2 (2018), 21:1–21:26. https://doi.org/10.1145/3133210

[46] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. 1996. Role-Based Access Control Models. *IEEE Computer* 29, 2 (1996), 38–47. https://doi.org/10.1109/2.485845

[47] Daniel Schlette, Florian Menges, Thomas Baumer, and Günther Pernul. 2020. Security Enumerations for Cyber-Physical Systems. In *Data and Applications Security and Privacy XXXIV - 34th Annual IFIP WG 11.3 Conference, DBSec 2020, Regensburg, Germany, June 25-26, 2020, Proceedings (Lecture Notes in Computer Science)*, Anoop Singhal and Jaideep Vaidya (Eds.), Vol. 12122. Springer, 64–76. https://doi.org/10.1007/978-3-030-49669-2_4

[48] Eric Spero, Michael P. Avera, Pierre E. Valdez, and Simon R. Goerger. 2014. Tradespace Exploration for the Engineering of Resilient Systems. In *Proceedings of the Conference on Systems Engineering Research, CSER 2014, Redondo Beach, CA, USA, March 20-22, 2014 (Procedia Computer Science)*, Azad M. Madni and Barry W. Boehm (Eds.), Vol. 28. Elsevier, 591–600. https://doi.org/10.1016/j.procs.2014.03.072

[49] Clay Stevens and Hamid Bagheri. 2020. Reducing run-time adaptation space via analysis of possible utility bounds. In *42nd International Conference on Software Engineering, ICSE '20, Virtual Event, USA, July 6-11, 2020*. ACM.

[50] Jouke Stoel, Tijs van der Storm, and Jurgen J. Vinju. 2019. AlleAlle: bounded relational model finding with unbounded data. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, Onward! 2019, Athens, Greece, October 23-24, 2019*, Hidehiko Masuhara and Tomas Petricek (Eds.). ACM, 46–61. https://doi.org/10.1145/3359591.3359726

[51] Allison Sullivan, Darko Marinov, and Sarfraz Khurshid. 2019. Solution Enumeration Abstraction: A Modeling Idiom to Enhance a Lightweight Formal Method. In *Formal Methods and Software Engineering - 21st International Conference on Formal Engineering Methods, ICFEM 2019, Shenzhen, China, November 5-9, 2019, Proceedings (Lecture Notes in Computer Science)*, Yamine Aït Ameur and Shengchao Qin (Eds.), Vol. 11852. Springer, 336–352. https://doi.org/10.1007/978-3-030-32409-4_21

[52] Maurice H. ter Beek, Kim G. Larsen, Dejan Ničković, and Tim A. C. Willemse. 2022. Formal Methods and Tools for Industrial Critical Systems. *Int. J. Softw. Tools Technol. Transf.* 24, 3 (jun 2022), 325–330. https://doi.org/10.1007/s10009-022-00660-4

[53] Dan Tofan, Matthias Galster, and Paris Avgeriou. 2013. Difficulty of Architectural Decisions - A Survey with Professional Architects. In *Proceedings of ECSA*. 192–199.

[54] Emina Torlak and Daniel Jackson. 2007. Kodkod: A Relational Model Finder. In *Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24 - April 1, 2007, Proceedings (Lecture Notes in Computer Science)*, Orna Grumberg and Michael Huth (Eds.), Vol. 4424. Springer, 632–647. https://doi.org/10.1007/978-3-540-71209-1_49

[55] Engin Uzuncaova and Sarfraz Khurshid. 2008. Constraint Prioritization for Efficient Analysis of Declarative Models. In *FM 2008: Formal Methods, 15th International Symposium on Formal Methods, Turku, Finland, May 26-30, 2008, Proceedings*. 310–325. https://doi.org/10.1007/978-3-540-68237-0_22

[56] Ru Wang, Anand Balu Nellippallil, Guoxin Wang, Yan Yan, Janet K. Allen, and Farrokh Mistree. 2018. Systematic design space exploration using a template-based ontological method. *Adv. Eng. Informatics* 36 (2018), 163–177. https://doi.org/10.1016/j.aei.2018.03.006