Web Semantics: Science, Services and Agents on the World Wide Web 72 (2022) 100694

Contents lists available at ScienceDirect

Web Semantics: Science, Services and Agents on the World Wide Web

journal homepage: www.elsevier.com/locate/websem

A multiplatform energy-aware OWL reasoner benchmarking framework

Floriano Scioscia, Ivano Bilenchi, Michele Ruta [*], Filippo Gramegna, Davide Loconte

Polytechnic University of Bari, Department of Electrical and Information Engineering, via E. Orabona 4, I-70125, Bari, Italy

ARTICLE INFO

Article history:
Received 15 January 2021
Received in revised form 6 December 2021
Accepted 13 December 2021
Available online 24 December 2021

Keywords:
Web Ontology Language
Automated reasoning
Benchmarking
Performance evaluation
Energy-aware computing

ABSTRACT

Performance evaluation is increasingly relevant for Web Ontology Language (OWL) reasoners, due to the expanding availability of knowledge corpuses on the Web, the growing variety of applications, and the rise to prominence of mobile and pervasive computing. Motivated mainly by the difficulty of comparing reasoning engines in the Semantic Web of Things (SWoT), this paper introduces evOWLuator, a novel approach and a multiplatform framework devised to be both flexible and expandable. It features integration of traditional and mobile/embedded engines as well as ontology dataset management, reasoning test execution, and report generation. A case study consisting of an experimental setting for time, memory peak and energy footprint evaluation with eight reasoners and four different platforms allows showcasing usage and validating features and usability of the tool.

1. Overview and motivation

One of the fundamental standards underpinning the Semantic Web is the Web Ontology Language (OWL), currently at version 2 [1]. It is used to create *ontologies*, *i.e.*, vocabularies endowed with a formal meaning which grounds terminological characterizations. OWL 2 semantics are based on the $\mathcal{SROIQ}$ Description Logic (DL), a fragment of First Order Logic (FOL).

Automated *reasoning* is a process to infer implicit knowledge from what has been explicitly declared in an ontology, and to answer specific queries. Inference engines (a.k.a. *reasoners*) typically provide their functionalities to other software components and systems as *services* through well-defined interfaces, such as Application Programming Interfaces (APIs) or protocols *e.g.*, *SPARQL* [2]. A wide range of reasoning services can be found in literature, and their computational properties have been extensively investigated and analogously, as research and development on Semantic Web has been very active for a long time, a large variety of reasoners still exist. A recent survey paper by the University of Manchester found 35 actively maintained OWL reasoners [3], and the authors' website contains an updated list[1] of 39 active projects and 35 apparently inactive ones.

When looking for the best tool for a particular application, besides functional requirements and platform compatibility, quantitative systematic analysis of performance and scalability becomes crucial. The selection of software components should rely as much as possible on rational processes based on quantitative data to prevent incorrect strategic decisions [4]. This is particularly true when dealing with performance and resource consumption evaluations, which may prevent running on a particular platform or implementing certain desired functionalities and thus lead to reduced acceptability and adoption of products and services.

This has motivated the creation of several OWL reasoning benchmarks and automated evaluation frameworks. Selecting an evaluation tool is by itself a non-trivial problem, depending on features like the types of collected performance metrics, platform compatibility, supported inference services, ease of reasoner integration, test automation capabilities. Furthermore, in latest years, the rise to prominence of mobile and ubiquitous computing has extended the field of application of knowledge representation and reasoning in non-traditional contexts. The integration of the Internet of Things (IoT) with knowledge representation and reasoning by means of Semantic Web technologies has been a recent ongoing effort, called *Semantic Web of Things* (SWoT) [5–8]. The goal is to permeate ordinary objects, events and environments with semantic annotations, which should be automatically generated, extracted, collected, organized and used by reasoning services. Possible applications are in smart device autonomous operations or context-aware user decision support. Devices based on Android and iOS are primary candidates for hosting SWoT reasoning engines, because they are endowed with non-negligible computational resources and a large array of embedded sensing

* Corresponding author.
E-mail addresses: floriano.scioscia@poliba.it (F. Scioscia), ivano.bilenchi@poliba.it (I. Bilenchi), michele.ruta@poliba.it (M. Ruta), filippo.gramegna@poliba.it (F. Gramegna), davide.loconte@poliba.it (D. Loconte).

1 List of Reasoners – OWL research at the University of Manchester, Internet Archive snapshot, April 2, 2019: https://web.archive.org/web/20190402132410/http://owl.cs.manchester.ac.uk/tools/list-of-reasoners/.

https://doi.org/10.1016/j.websem.2021.100694
1570-8268/© 2021 Elsevier B.V. All rights reserved.

and communications capabilities. They allow gathering significant amounts of heterogeneous context-sensitive information. In latest years, further targeted device platforms include single-board computers – useful in Mobile Edge Computing [9] for analyzing information at the edge of local area networks – and even embedded devices and Programmable Logic Controllers (PLCs) [10].

SWoT contexts are affected by several additional challenges with respect to traditional Web-oriented computing infrastructures and systems, including:

- high device volatility, due to mobility and battery constraints;
- high information volatility, due to the *velocity* of sensor-generated Big Data streams [11];
- platform heterogeneity, depending not only on the wide hardware and software variety of device platforms, but also on the different – and often incompatible – IoT frameworks for information extraction and device management, which makes cross-platform interoperability both necessary and difficult;
- low computational and energy resources, requiring extremely careful optimization of CPU, memory and energy usage.

Due to the above motivations, reasoning engines and benchmarks for conventional computing architectures in the Semantic Web cannot be promptly leveraged in SWoT contexts. Porting existing tools is often unfeasible because of deep hardware and software platform differences, and even when possible it is a major effort which does not always pay off in terms of performance. Specifically, energy consumption has never been considered in reasoner design for conventional computing architectures. While several Java-based Semantic Web reasoners have been ported to Android [12], the landscape is almost completely barren for iOS devices [13] and embedded real-time operating systems. Furthermore, required inference services are quite different between typical Semantic Web and SWoT scenarios: in the former, complex queries on large Knowledge Bases (KBs) with moderate-to-high expressiveness and batch jobs are the norm, whereas the latter includes a wide range of specialized use cases, predominantly based on smaller KBs with low-to-moderate expressiveness and quick on-the-fly queries. For the above reasons, at the moment SWoT-oriented reasoners are sorely lacking rigorous evaluation methodologies, benchmarks and software frameworks, which unfortunately have little overlap with existing approaches. To the best of our knowledge all the available OWL reasoner evaluation frameworks and tools – including all the ones reported in Section 5 – do not provide a fully satisfactory solution in terms of flexible support of both standard and non-standard reasoning services, wide platform compatibility and energy consumption estimation in a single package.

This paper presents EVOWLUATOR,[2] a novel multi-platform framework for the evaluation of OWL reasoning tasks. Original contributions include:

- high flexibility, expandability and scalability by means of peculiar architectural choices, differentiating the proposal from the state of the art;
- unrestricted support of ontology corpuses for benchmarking;
- extensible list of supported reasoning services, including *ontology classification* and *consistency check*, as well as the *semantic matchmaking* non-standard task based on non-monotonic inference, which is particularly useful in SWoT scenarios for resource discovery and event detection [14];
- capability to deploy tests either locally or to remote devices, thus enabling their orchestration on mobile and embedded platforms;
- evaluation of reasoning correctness, turnaround time, memory usage and energy footprint;
- a plug-in architecture supporting the integration of additional target reasoners and platforms;
- generation of result visualizations as interactive plots, making the framework particularly useful in research activities.

In order to exemplify best practices and provide practitioner insights concerning architectures, techniques, management and orchestration of ontology artifacts and reasoning tools for the successful preparation and execution of benchmarks, a relatively small experimental campaign is reported as a case study, including 8 reasoners, $\simeq$1300 low-expressiveness ontologies of various size and 20 fairly large high-expressiveness ontologies. This allows exploring, testing and validating the framework's functionalities to a significant extent in a non-trivial case.

The remainder of the paper is as follows: Section 2 recalls useful details about OWL ontologies and specific inference tasks. The case study is in Section 3, comprising both framework usage guidelines and relevant results. Section 4 describes EVOWLUATOR's architecture in detail. Related work is discussed in Section 5, before conclusion.

2. Preliminaries

In the semantics of OWL, basic elements include: *classes* (a.k.a. *concepts* in DL jargon), representing sets of objects; *object properties* (a.k.a. *roles*), linking pairs of objects; *data properties* (a.k.a. *functional roles on concrete domains*), linking objects with data values (a.k.a. *literals*); *individuals* (a.k.a. *instances*), representing specific objects. These elements can be combined using *constructors* to form *class expressions*, which can be used in sets of *inclusion assertions* and *definitions* called *TBoxes* (Terminological Boxes), which impose restrictions on possible interpretations according to the knowledge elicited for a given *domain*. A TBox represents a formal shared specification of a conceptualization and contains knowledge referring to a set of possible worlds, whereas an *ABox* (Assertion Box) contains assertions about individuals and their relations referring to a specific world state or problem [15]. In DL systems, the term "ontology" refers specifically to a TBox, while the union of a TBox and ABox form a Knowledge Base; however, among practitioners of Semantic Web technologies "ontology" is often used as a synonym of "Knowledge Base", and from now on for convenience this is done here, too.

Reasoning services elicit implicit knowledge from an ontology replying to given requests (queries). Usually, a sharp categorization distinguishes *standard* inferences (including class *satisfiability* and *subsumption*, ontology *classification* and *consistency check* along with instance *retrieval* and *realization* [16]) and a large variety of *non-standard* ones, specifically adopted in non-conventional use cases and special-purpose reasoners. As an example, *JustBench* [17] enables benchmarking the *justification* correctness check. Analogously, EVOWLUATOR supports the non-standard *Concept Abduction* (CA) and *Concept Contraction* (CC) [14] inference services, which are particularly useful in SWoT-oriented matchmaking and negotiation scenarios as an extension and an explanation of (failed) subsumption and satisfiability checks, respectively.

In order to clarify the benchmarking framework outline in Section 4, it is useful to recall the basics of the matchmaking scheme described in [14], which relies on CC and CA. Let us consider a TBox $\mathcal{T}$ and R, S two concepts – representing a *request*

2 EVOWLUATOR project repository: https://github.com/sisinflab-swot/evowluator.

and a *resource*, respectively – both satisfiable in $\mathcal{T}$. If $S \sqcap R$ is unsatisfiable in $\mathcal{T}$, CC determines which part of R clashes with S. By retracting only conflicting requirements G (for *Give up*) from R, an expression K (for *Keep*) remains, representing a contracted version of the original request. The solution G to Contraction explains "why" the conjunction of R and S is not satisfiable. Conversely, if R and S do not clash, but subsumption $\mathcal{T} \models S \sqsubseteq R$ does not hold – *i.e.*, characteristics of the resource do not cover the request fully –, CA determines the part H (for *Hypothesis*) of R that is requested though not specified in S, providing an explanation for missed subsumption.

The matchmaking service supported by EVOWLUATOR aims to rank a set of resources $\mathcal{S} = \{S_i \mid i = 1, \ldots, n\}$ by semantic similarity with a common request R. This 1-to-n problem is solved through n 1-to-1 comparisons. Given R and S_i, a preliminary *compatibility check* is carried out, which fails if $S_i \sqcap R$ is unsatisfiable in $\mathcal{T}$. If they are not compatible, CC is first computed in order to "fit" the request to the available resource, then CA is performed between the resulting K_i and S_i; otherwise, if they are compatible, CA is directly computed between R and S_i. As an example, consider the following concept expressions:

$$S_1 \rightarrow A \sqcap B \sqcap (\geq 3P)$$
$$S_2 \rightarrow A \sqcap \neg B \sqcap (\geq 4P)$$
$$R \rightarrow B \sqcap C \sqcap (\geq 2P)$$

S_1 is compatible with R though it is not subsumed by it, so CA can be performed in order to find out why subsumption does not hold: $CA(S_1, R) = C$. On the other hand, S_2 is not compatible with R, so we can use CC in order to "split" R into a $\langle G, K \rangle$ pair containing the incompatible and compatible parts of R, respectively: $CC(S_2, R) = \langle B, C \sqcap (\geq 2P) \rangle$. Then CA can be computed for the compatible part: $CA(C \sqcap (\geq 2P), R) = C$.

For DLs which admit a normal form for concept expressions, a metric space with a *norm* operator $\| \cdot \|$ can be defined, as is the case, *e.g.*, for the Conjunctive Normal Form (CNF) in the $\mathcal{ALN}$ (Attributive Language with unqualified Number restrictions) DL [14]. In such case, the norm of G and H represents a semantic distance *penalty* for CC and CA, respectively, which can be used in the matchmaking framework to rank resources with respect to a given request. In the above example, S_1 will be ranked better than S_2 w.r.t. R, since both resources have the same CA penalty but the latter also has a CC penalty.

3. Case study: benchmarking ontology classification and consistency

EVOWLUATOR is a multiplatform software framework devised to assess the correctness, performance and energy footprint of inference services exposed by OWL reasoners. In order to promote its adoption in both academic and industrial contexts, it is released under the Eclipse Public License (EPL) version 2.0.[3]

A small experimental campaign has been carried out to validate EVOWLUATOR's effectiveness and features as well as to exemplify its usage. Correctness, performance and energy footprints of the *ontology classification* and *consistency* inference services implemented by some state-of-the-art OWL reasoners have been evaluated on both desktop and mobile devices. Obtained results are available at a permanent URL[4] for the sake of reproducibility.

In the following subsections the testbed, EVOWLUATOR setup and usage, and obtained results are reported. The main purpose is assessing the framework capabilities as a benchmarking platform, therefore tests refer to a subset of reasoners and datasets smaller than other initiatives (reported in Section 5), mostly oriented to systems comparison and competition.

[3] Eclipse Public License 2.0: https://www.eclipse.org/legal/epl-2.0/.

[4] https://zenodo.org/record/3473133.

3.1. Testbed, reasoners and datasets

Desktop tests have been performed on a *Linux* workstation[5] and a *Mac Mini (2014)*,[6] while mobile experiments have been carried out on an *Apple iPhone 7*[7] and a *HTC/Google Nexus 9* tablet.[8] Tested desktop reasoners include: *Fact++* (version 1.6.5) [18], *HermiT* (1.3.8) [19], *Konclude* (0.6.2-544) [20], *Mini-ME* (2.0) [14], *Mini-ME Swift* (1.0) [13] and *TrOWL* (1.5) [21]. Mini-ME and Mini-ME Swift have also been used for tests on Android and iOS, respectively. Additionally, the *JFact* (1.2.1), *HermiT* (1.3.8) and *Pellet* (2.3.1) [22] Android ports from [12] have been evaluated.

Tests have been carried out on the following datasets:

- **ORE 2014**: 1398 ontologies selected from the *2014 OWL Reasoner Evaluation Workshop* competition dataset,[9] considering only those having at most $\mathcal{ALN}$ as reference expressiveness. This allows demonstrating the evaluation of mobile reasoners such as Mini-ME, which do not support more expressive languages;
- **ORE 2014 Energy**: it consists of the 50 largest ontologies from the ORE 2014 set (average size 1781.28 ± 14063.8 kB, minimum 11.44 kB, maximum 291.33 MB in functional syntax), and it has been employed to carry out energy consumption tests on macOS. The rationale behind running these tests only on large ontologies is that software energy profilers such as *PowerMetrics*[10] and *PowerTOP*[11] are accurate for processes whose runtime is sufficiently long. In particular, they are unable to provide any data at all for processes that spawn and terminate before any power sample is collected, regardless of how energy-intensive they may be.
- **BioPortal**: a dataset composed of 20 ontologies from BioPortal[12] [23] with no restrictions on DL expressiveness (average size 130.79 ± 104.57 MB, minimum 27.92 MB, maximum 450.71 MB in functional syntax). BioPortal has been chosen because life sciences ontologies are among the largest as well as the best known by practitioners of Semantic Web technologies. Ontologies have been selected by taking the 100 largest ones in BioPortal, classifying them via four high-expressiveness reasoners (Konclude, Fact++, HermiT and TrOWL), and selecting only those for which all reasoners returned correct and complete inferences within 2 h. This dataset has been used to demonstrate energy profiling capabilities on both Linux and macOS.

3.2. Setup

Experiments setup follows the procedure reported in detail in EVOWLUATOR's documentation. The adopted configuration is available online[13] as an example.

[5] AMD Ryzen 5 3600 at 3.6 GHz, 32 GB DDR4 RAM at 3000 MT/s, 1 TB SSD, Ubuntu 18.04 LTS x64.

[6] Intel i7 4578u dual-core CPU at 3.0 GHz, 16 GB DDR3 RAM at 1600 MT/s, 1 TB HDD + 128 GB SSD (Fusion Drive), macOS Mojave 10.14.5.

[7] Apple A10 CPU (2 high-performance cores at 2.34 GHz and 2 low-energy cores), 2 GB LPDDR4 RAM, 32 GB flash storage, iOS 10.1.1.

[8] Nvidia Tegra K1 dual-core CPU at 2.3 GHz, 2 GB LPDDR3 RAM at 1600 MT/s, 32 GB flash memory, Android 7.1.1 Nougat, patch level 5 October 2017.

[9] http://dl.kr.org/ore2014.

[10] PowerMetrics: https://developer.apple.com/library/archive/documentation/Performance/Conceptual/power_efficiency_guidelines_osx/PrioritizeWorkAtTheTaskLevel.html.

[11] PowerTOP: https://01.org/powertop.

[12] https://bioportal.bioontology.org.

[13] http://swot.sisinflab.poliba.it/evowluator/downloads/config.7z.

Datasets. After installing EVOWLUATOR on both desktop machines in an `evowluator` base directory, the aforementioned datasets have been set up by moving ontologies into appropriate subdirectories of the `data` directory in the install path. Each dataset is expected to have a root directory, whose name is used as identifier, and a subdirectory for each supported OWL syntax, which in turn must contain ontology files.[14]

Desktop reasoners. The next step has concerned the integration of reasoners, which must be configured by writing Python modules implementing the `Reasoner` interface provided by the framework and placing them into the `reasoners` subdirectory under the install path. For Java-based reasoners supporting the OWL API, a wrapper has been created to expose the classification and consistency inference tasks, ensuring all reasoners have the same command line interface. This has covered Fact++, HermiT, TrOWL and Mini-ME. The Python side of the integration has then been accomplished by writing a single `Reasoner` subclass for all OWL API reasoners, further subclassed to configure individual reasoner metadata. Mini-ME Swift and Konclude come with a built-in command line interface, instead, so providing befitting `Reasoner` subclasses has been enough to integrate them.

iOS reasoners. After installing Xcode and its command line tools on the host machine, supporting iOS reasoners like Mini-ME Swift has required creating a `XCTest` project (as per XCode documentation) and wrapping reasoning task invocation in separate methods of an `XCTestCase` subclass. EVOWLUATOR provides the `IOSReasoner` template class for the Python side of the configuration, which must be subclassed to specify the appropriate arguments for launching inferences via `xcodebuild`. Ontologies have been uploaded to the target device through the *copy bundle resources* Xcode build phase, though this is not strictly needed, as the app might retrieve the necessary ontologies by other means.

Android reasoners. The `adb` tool is required on the host computer to enable communication with the target Android device.[15] Wrapper Android apps – which have been implemented for Mini-ME, JFact, HermiT and Pellet – basically respond to specific intents[16] and start the corresponding reasoning tasks. The Python side of the configuration has involved subclassing the `AndroidReasoner` template provided by EVOWLUATOR, which allows the framework to invoke reasoner apps via `adb`. Ontologies have been uploaded to the external memory of the device prior to starting the tests.

Section 4.2 discusses on architectural details of the aforementioned interfaces and template classes to be extended for framework configuration.

3.3. Test execution and results generation

After setup, running tests has just involved invoking the `evowluate` executable with appropriate arguments. The following subcommands are supported:

- `classification | consistency | matchmaking`: runs the evaluation for the specified inference task, and outputs outcomes into the `results` directory;
- `info`: prints information about configured reasoners and datasets;
- `visualize`: generates high-level statistics and plots from the output of a previous evaluation;
- `convert`: converts a dataset to the specified OWL syntax, using the `owltool` helper.

The `-h` flag can be used to get further information about each subcommand; for instance, `./evowluate classification -h` lists all the options related to the classification task.

Each inference task can be evaluated in one of three possible modes via the `-m` flag. In particular:

- *correctness*: checks the validity of reasoners' output, using an oracle. The test oracle is the first reasoner specified in the list following the `-r` flag;
- *performance*: collects statistics about performance, in terms of turnaround time and maximum memory usage;
- *energy*: computes an estimate of the energy drained by the inference task. This mode requires specifying the class name of the energy probe the framework should use through the `-e` flag.

The user can set further options with specific flags, or leave default values: `-d` for the target dataset (default: the first dataset in the `data` directory in lexicographic order); `-r` for the list of reasoners to be used (default: all configured ones); `-n` for the number of iterations for each test (default: 5); `-t` for the timeout imposed on each reasoner, in seconds (default: 1200); `-s` for the reference OWL syntax (default: the preferred syntax for each reasoner).

The `visualize` subcommand is used to generate summaries and plots out of evaluation results. It requires the path of the directory containing the results of a previous evaluation and supports many optional flags, mainly aimed at a fine-tuning plot appearance and not reported here for the sake of brevity. Finally, the `convert` subcommand can be used to convert existing datasets into additional syntaxes, to be selected among `dl`, `functional`, `krss`, `krss2`, `manchester`, `obo`, `owlxml`, `rdfxml` and `turtle`.

3.4. Results

Correctness. Correctness has been checked on the whole ORE 2014 dataset using Konclude as test oracle, since the latest OWL reasoner competition [24] reported it as the most reliable with regards to ontology classification and consistency. The outcomes are illustrated in Fig. 1, which shows the number of correct and incorrect results for each reasoner with respect to the test oracle.

Table 1
Summary of classification correctness tests.

Reasoner	Correct	Incorrect	Timeout (s)	Error	Ratio
Fact++	1393	1	0	4	1.00
HermiT	1398	0	0	0	1.00
Mini-ME	1168	222	8	0	0.84
Mini-ME Swift	1364	34	0	0	0.98
TrOWL	1396	1	1	0	1.00

Table 2
Summary of consistency correctness tests.

Reasoner	Correct	Incorrect	Timeout (s)	Error	Ratio
Fact++	1394	0	0	4	1.00
HermiT	1398	0	0	0	1.00
Mini-ME	1315	75	8	0	0.94
Mini-ME Swift	1387	11	0	0	0.99
TrOWL	1398	0	0	0	1.00

[14] Missing ontology formats, as needed by some of the reasoners, can be generated using the `convert` subcommand of `evowluate`, the main tool in the framework. As an example, the ORE 2014 dataset is only available in functional OWL syntax, though Mini-ME Swift requires ontologies in RDF/XML format, so they have been generated via the command:

```
evowluate convert -d ore2014 -s rdfxml.
```

[15] *USB debugging* must also be enabled via the Settings app of the target device, by accessing the hidden *developer menu*, as explained in https://developer.android.com/studio/debug/dev-options.

[16] Android intents: https://developer.android.com/reference/android/content/Intent.

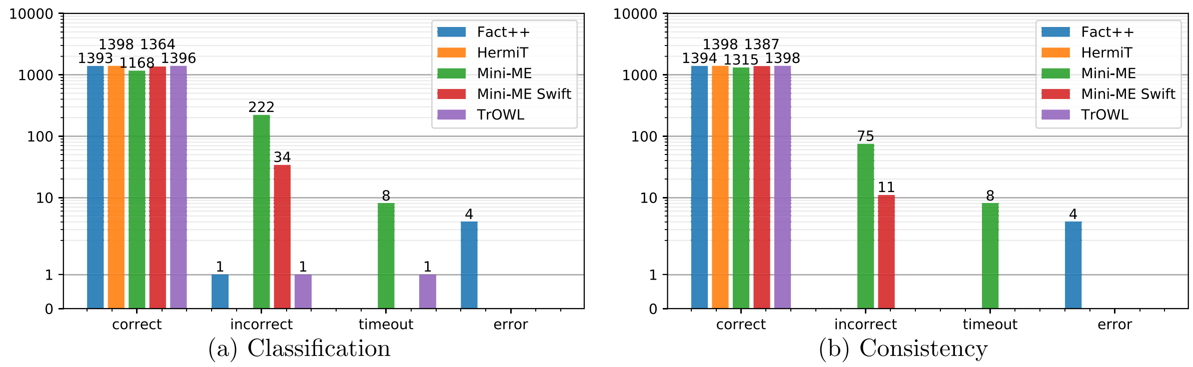


Fig. 1. Correctness results.

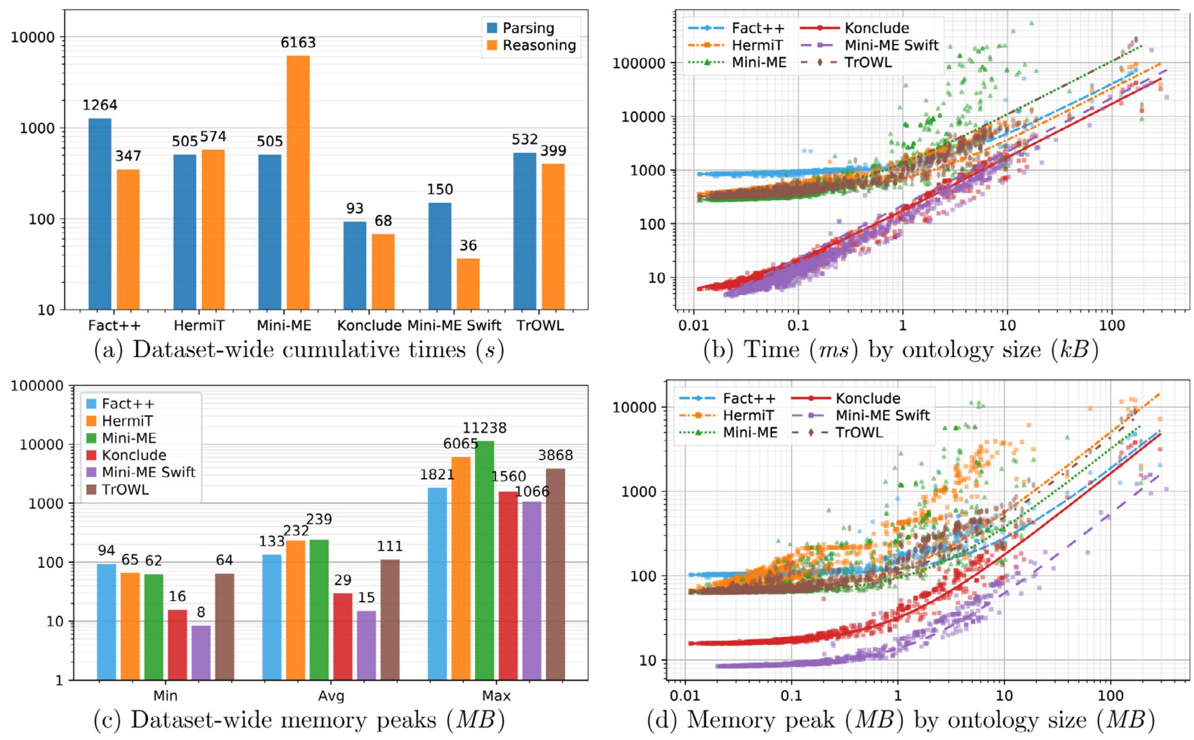


Fig. 2. Classification performance tests on desktop.

Table 3
Summary of classification performance tests on desktop.

Reasoner	Parsing time (s)	Reasoning time (s)	Total time (s)	Min mem. peak (MB)	Avg mem. peak (MB)	Max mem. peak (MB)
Fact++	1247.31	347.48	1594.79	93.57	131.00	1728.90
HermiT	491.35	573.57	1064.92	65.45	227.37	6064.55
Konclude	76.28	67.64	143.91	15.53	27.39	455.15
Mini-ME	491.44	6162.98	6654.41	62.49	233.34	11237.72
Mini-ME Swift	121.37	36.40	157.77	8.41	13.91	189.50
TrOWL	516.93	399.39	916.32	63.88	105.78	1353.89

The plot also reports on the times reasoners have hit the imposed timeout, and the number of runtime errors. The same data is also reported in Tables 1 and 2, which further include a correctness ratio, computed as the number of correct results over the number of ontologies in the dataset. It can be noted Mini-ME and Mini-ME Swift exhibit lower ratios than the other reasoners: incorrect results for Mini-ME and Mini-ME Swift are due to unsupported constructs in the ontologies; the former has additional timeouts on the largest ontologies of the dataset.

Desktop performance. Performance evaluation metrics refer to the ontologies from the ORE 2014 dataset which all the above reasoners have classified correctly within the timeout on the macOS testbed. Results for classification and consistency are pictured in Figs. 2 and 3, respectively, evidencing EVOWLUATOR's capability to generate histograms and scatterplots. In detail:

- Figs. 2(a) and 3(a) show histogram plots of dataset-wide cumulative parsing and reasoning times in seconds.
- Figs. 2(b) and 3(b) depict time as a function of ontology size.

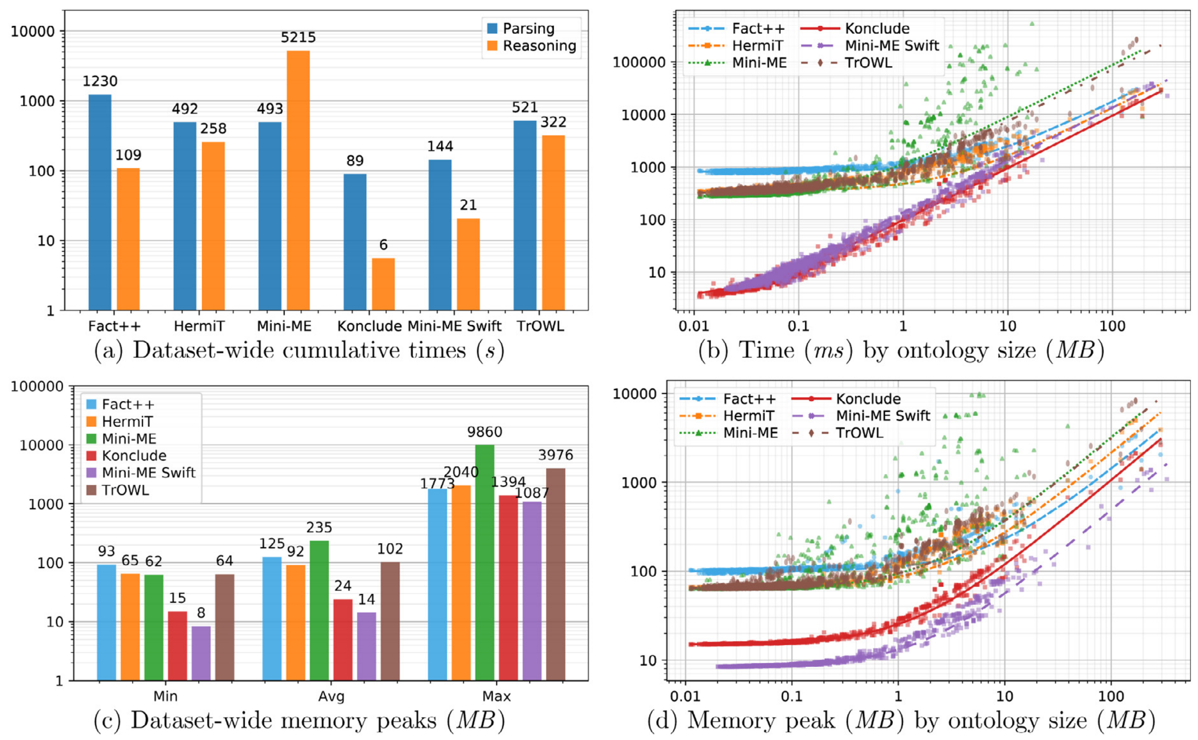


Fig. 3. Consistency performance tests on desktop.

Table 4
Summary of consistency performance tests on desktop.

Reasoner	Parsing time (s)	Reasoning time (s)	Total time (s)	Min mem. peak (MB)	Avg mem. peak (MB)	Max mem. peak (MB)
Fact++	1212.18	108.55	1320.73	92.98	122.94	1718.54
HermiT	477.92	258.24	736.16	64.95	89.34	649.09
Konclude	72.57	5.58	78.15	14.96	22.19	245.55
Mini-ME	479.05	5214.83	5693.88	62.38	229.72	9859.68
Mini-ME Swift	114.93	20.69	135.62	8.36	13.16	167.49
TrOWL	506.20	321.51	827.71	63.71	97.32	1278.56

- Figs. 2(c) and 3(c) illustrate dataset-wide minimum, average and maximum memory peaks for each reasoner.
- Figs. 2(d) and 3(d) plot memory peak as a function of ontology size.

Aggregated results output by the framework are summarized in Tables 3 and 4, showing total parsing and reasoning time, as well as dataset-wide minimum, average and maximum memory peak for each reasoner.

Mobile performance. Similar plots are shown for mobile tests, sketching performance metrics of Android reasoners (Figs. 4 and 5) and Mini-ME Swift running on iOS (Figs. 6 and 7). Both desktop and mobile outcomes are in line with previous experimental campaigns [13].

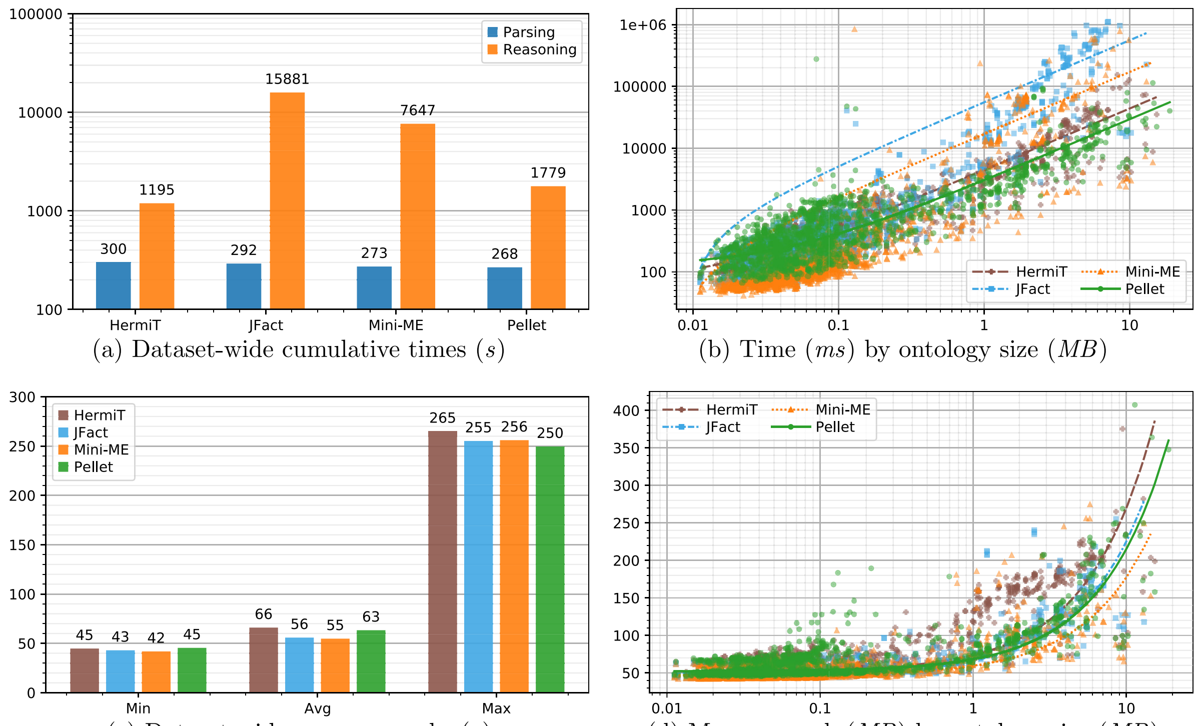
It can also be noticed how evOWLuator uses *Matplotlib* to automatically adapt and differentiate graph elements (size and color) of reports depending on the number of tested reasoners. All figures in this section have been exported to Portable Document Format (PDF) and integrated with no modification in the LaTeX project of this manuscript; likewise they can be formatted to Scalable Vector Graphics (SVG) or Portable Network Graphics (PNG) for Web publishing.

ORE 2014 Energy footprint. These tests measure the energy consumption of reasoning tasks on the `ORE 2014 Energy` dataset. All tests have been executed on the macOS testbed. Results are shown in Fig. 8; it is important to recall they represent energy usage, therefore lower scores are better. In particular:

- Figs. 8(a) and 8(c) recall dataset-wide minimum, average and maximum energy footprint for each reasoner.
- Figs. 8(b) and 8(d) plot energy footprint as a function of the ontology size.
- Tables 5 and 6 summarize energy evaluation results: they provide the same information as Figs. 8(a) and 8(b), though in tabular form.

It should be noticed how Fact++ and HermiT have a significantly smaller energy footprint for consistency than classification, while for the other reasoners the two scores are closer.

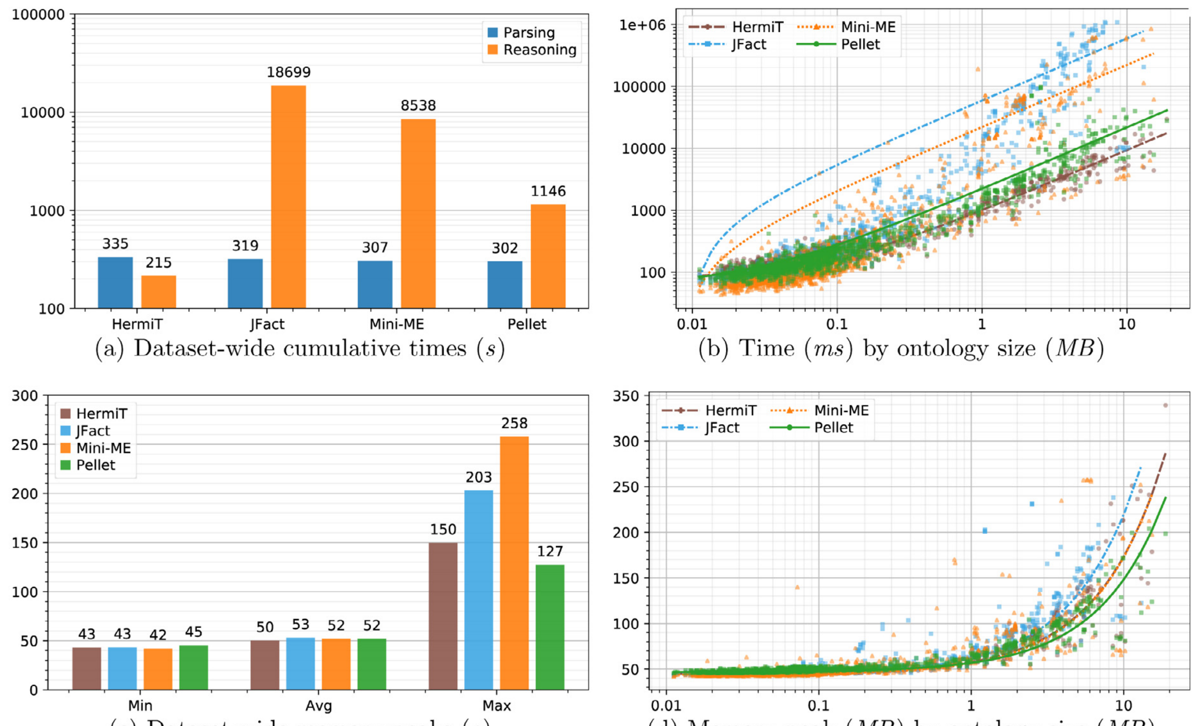
The availability of results in CSV format facilitates further processing through the *pandas* Python library. As an example, Table 7 has been created by computing the time–energy, memory–energy and time–memory Pearson correlation coefficients for each reasoner for the classification task, derived on the `ORE 2014 Energy` dataset. They have been computed as follows: average results output by performance and energy evaluations are loaded and merged through the `DataFrame.merge()` method; pairwise correlation between columns is then computed via the



(a) Dataset-wide cumulative times (s) (b) Time (ms) by ontology size (MB)

(c) Dataset-wide memory peaks (s) (d) Memory peak (MB) by ontology size (MB)

Fig. 4. Classification performance tests on Android.



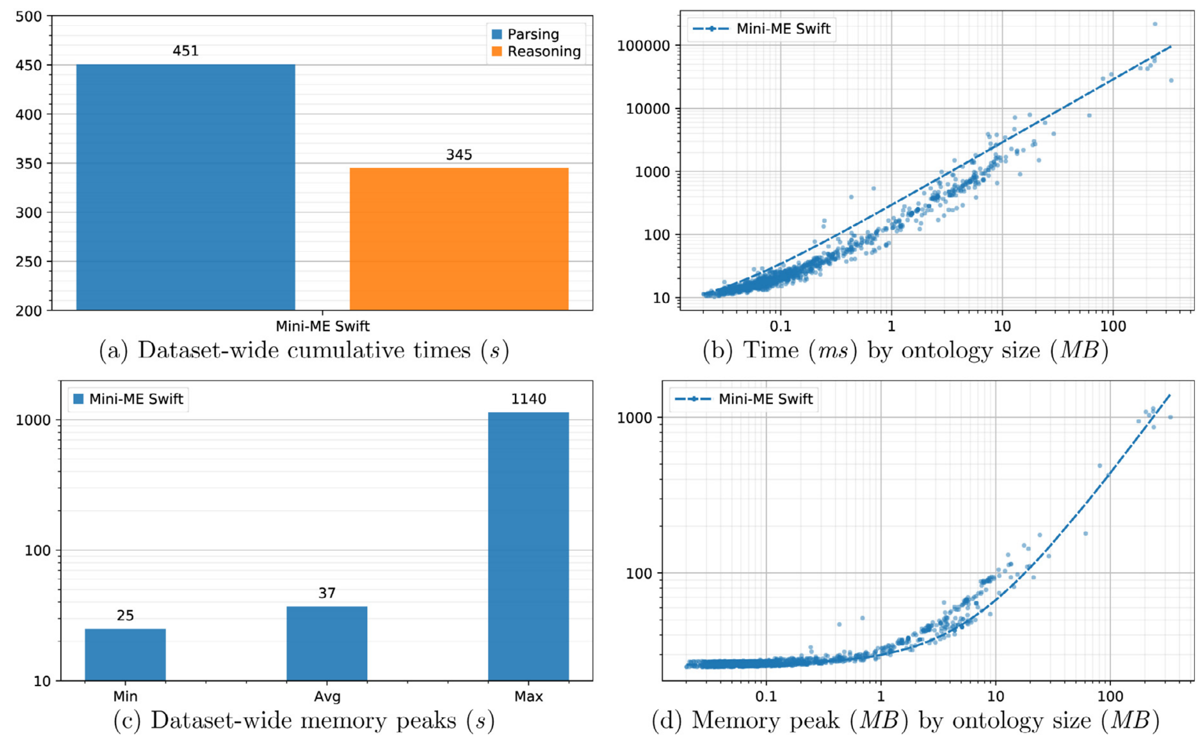
(a) Dataset-wide cumulative times (s) (b) Time (ms) by ontology size (MB)

(c) Dataset-wide memory peaks (s) (d) Memory peak (MB) by ontology size (MB)

Fig. 5. Consistency performance tests on Android.

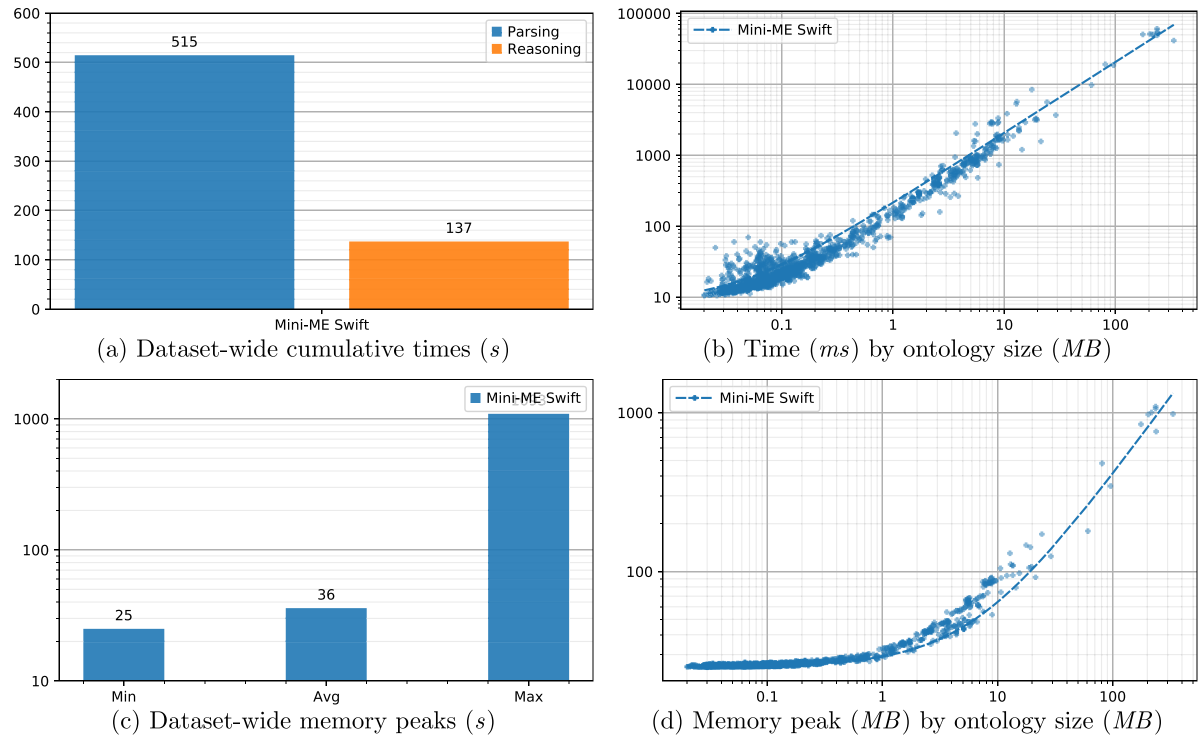
`DataFrame.corr()` method.[17] Values indicate a strong linear correlation between energy footprint and time for all reasoners, in accordance with [25] but partial disagreement with [26], as discussed in Section 5. Linear correlation is also found between energy and memory usage in the majority of tested engines, with the exceptions of Fact++ and Mini-ME, whose behavior calls for further investigation through specific experiments.

[17] Pandas DataFrame documentation: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html.



(a) Dataset-wide cumulative times (s) (b) Time (ms) by ontology size (MB) (c) Dataset-wide memory peaks (s) (d) Memory peak (MB) by ontology size (MB)

Fig. 6. Classification performance tests on iOS.

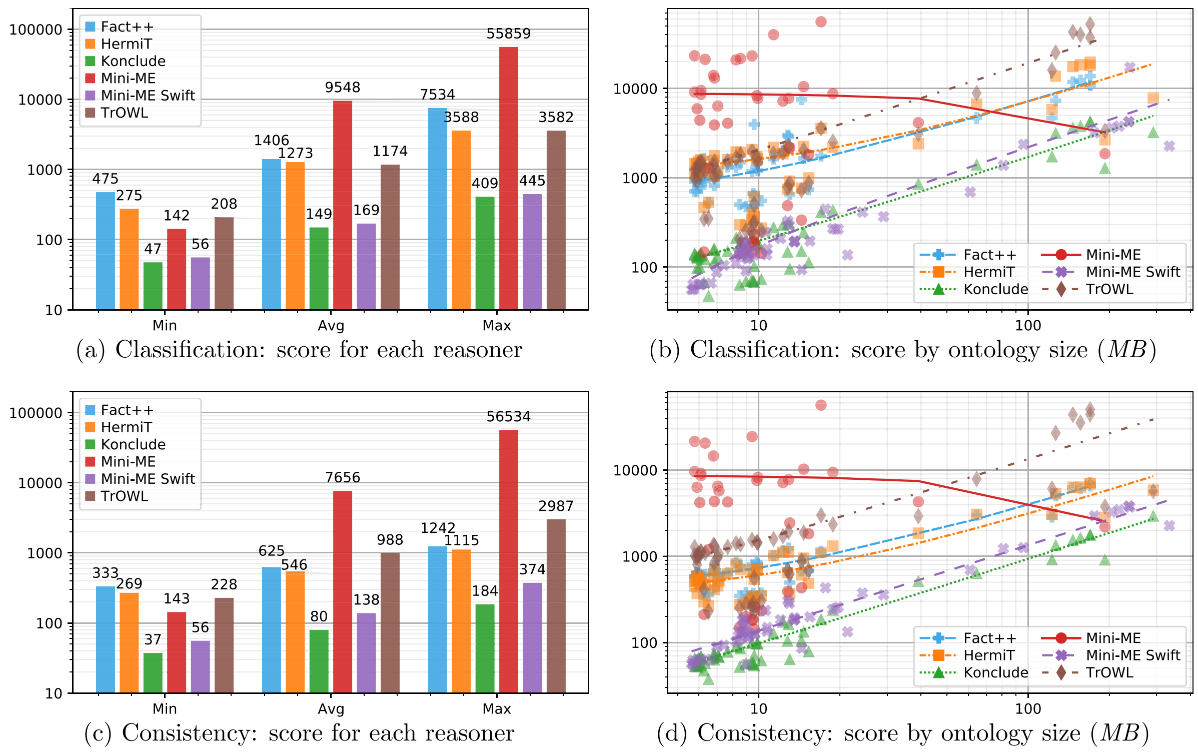


(a) Dataset-wide cumulative times (s) (b) Time (ms) by ontology size (MB) (c) Dataset-wide memory peaks (s) (d) Memory peak (MB) by ontology size (MB)

Fig. 7. Consistency performance tests on iOS.

BioPortal Energy Footprint. In order to assess EVOWLUATOR features on multiple platforms and on very large and expressive ontologies, an additional experimental session has been carried out, measuring energy footprint on both Linux and macOS on the `BioPortal Energy` dataset. Mini-ME Swift has been excluded from this test as its supported expressiveness is limited to $\mathcal{ALN}$. Results are summarized as follows:

- Fig. 9 reports on energy scores measured on Linux via `powertop`;
- Tables 8 and 9 refer to Linux tests in a tabular form;
- Fig. 10 recalls energy scores measured on macOS through `powermetrics`;
- Tables 10 and 11 reports macOS tests results in a tabular form;



(a) Classification: score for each reasoner   (b) Classification: score by ontology size (MB)

(c) Consistency: score for each reasoner   (d) Consistency: score by ontology size (MB)

Fig. 8. ORE 2014 — Energy footprint tests on macOS.

Table 5
ORE 2014 — Summary of classification energy footprint tests on macOS.

Reasoner	Min energy	Avg energy	Max energy
Fact++	475.44	1406.19	7533.51
HermiT	274.50	1273.30	3588.50
Konclude	47.48	149.18	408.60
Mini-ME	141.93	9547.86	55859.39
Mini-ME Swift	55.74	169.42	445.29
TrOWL	208.21	1174.35	3582.45

Table 6
ORE 2014 — Summary of consistency energy footprint tests on macOS.

Reasoner	Min energy	Avg energy	Max energy
Fact++	332.82	624.91	1241.54
HermiT	269.42	545.51	1115.08
Konclude	37.33	79.82	184.28
Mini-ME	143.05	7655.57	56533.58
Mini-ME Swift	55.75	137.92	373.81
TrOWL	228.31	987.98	2987.28

Table 7
ORE 2014 — Correlation between time, memory peak and energy footprint score on macOS.

Reasoner	Energy–Time	Energy–Memory	Time–Memory
Fact++	0.98	0.27	0.19
HermiT	0.99	0.96	0.97
Konclude	1.00	0.99	0.99
Mini-ME	1.00	0.28	0.24
Mini-ME Swift	1.00	0.93	0.93
TrOWL	0.99	0.97	0.98

Table 8
BioPortal — Summary of classification energy footprint tests on Linux.

Reasoner	Min energy	Avg energy	Max energy
Fact++	23.87	1606.24	6747.2
HermiT	32.65	236.07	1244.58
Konclude	3.61	21.78	56.34
TrOWL	30.7	80.47	171.51

Table 9
BioPortal — Summary of consistency energy footprint tests on Linux.

Reasoner	Min energy	Avg energy	Max energy
Fact++	21.77	45.8	94.99
HermiT	20.51	62.66	133.28
Konclude	2.76	16.75	51.4
TrOWL	26.5	79.65	162.73

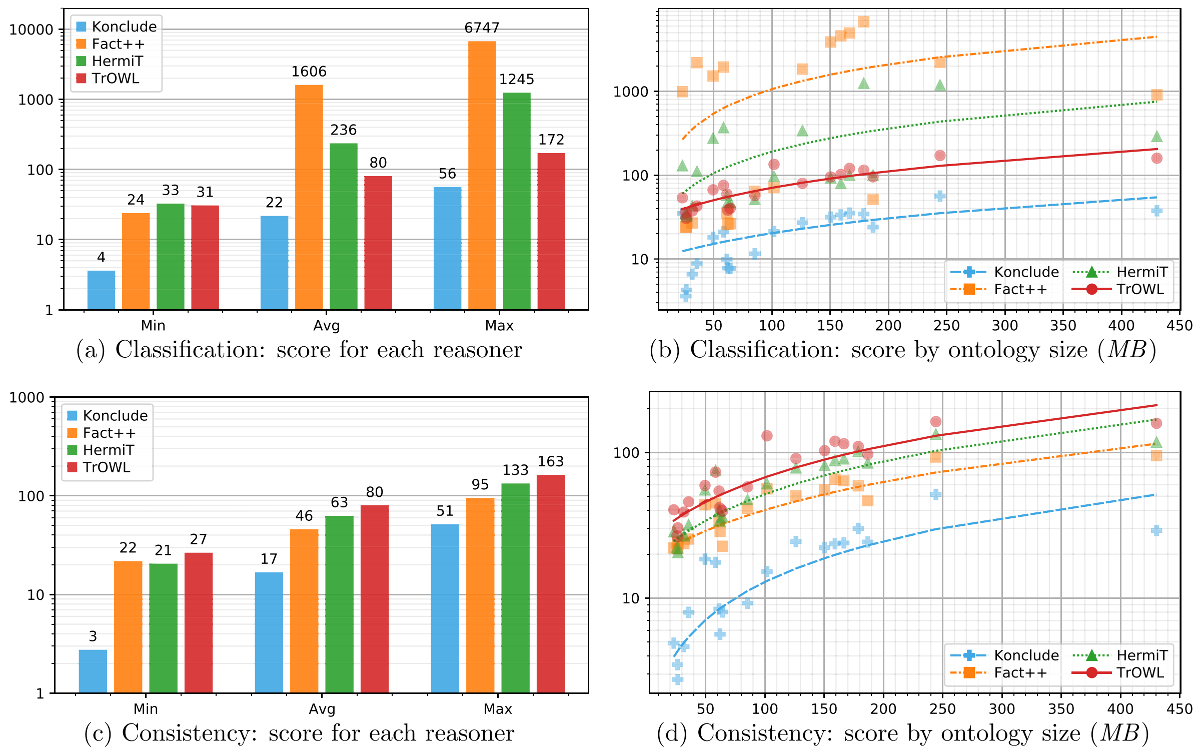
Table 10
BioPortal dataset — Summary of classification energy footprint tests on macOS.

Reasoner	Min energy	Avg energy	Max energy
Fact++	1472.49	131134.95	563359.07
HermiT	2524.55	17081.62	95200.06
Konclude	243.23	1703.41	4976.11
TrOWL	2442.32	6483.57	13857.09

Table 11
BioPortal dataset — Summary of consistency energy footprint tests on macOS.

Reasoner	Min energy	Avg energy	Max energy
Fact++	1282.61	2949.97	6136.69
HermiT	1436.54	4305.14	9269.16
Konclude	243.23	1703.41	4976.11
TrOWL	1991.29	5736.65	12179.98

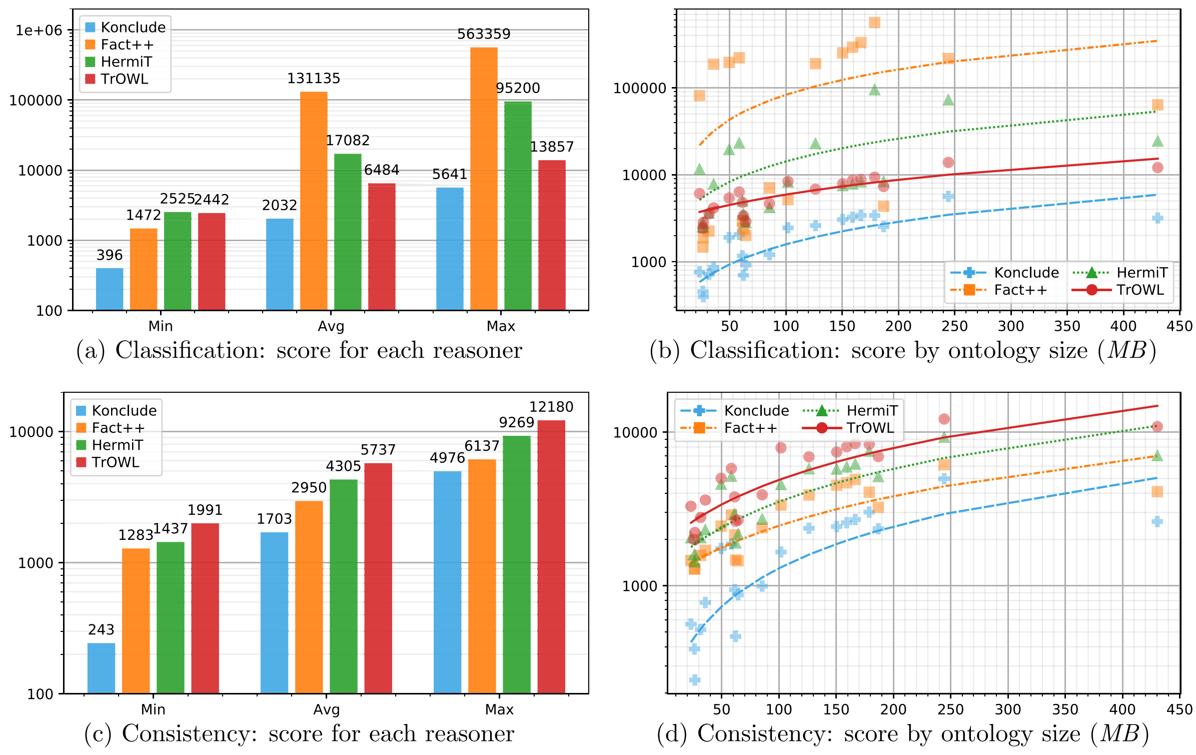
Also in this case Fact++ and HermiT exhibit a significantly smaller energy footprint for consistency than classification, while each of the other reasoners behave more similarly across the two inferences. Tables 12 and 13 are generated analogously to Table 7 from the previous test case, reporting the time–energy, memory–energy and time–memory Pearson correlation coefficients for each reasoner. They are quite consistent, even referring to different operating systems and energy profilers. In fact, the



(a) Classification: score for each reasoner (b) Classification: score by ontology size (MB)

(c) Consistency: score for each reasoner (d) Consistency: score by ontology size (MB)

Fig. 9. BioPortal Energy footprint tests on Linux desktop.



(a) Classification: score for each reasoner (b) Classification: score by ontology size (MB)

(c) Consistency: score for each reasoner (d) Consistency: score by ontology size (MB)

Fig. 10. BioPortal Energy footprint tests on Mac Mini.

correlation between *powertop* scores on Linux and *powermetrics* measurements on macOS, reported in Table 14, is quite high. Comparing Table 7 with Tables 12–13, it appears that for larger ontologies HermiT has correlation values more similar to those of Fact++. This may mean lower energy–time and energy-memory correlations are a byproduct of lower time–memory correlation, which depends on the interplay between inference algorithms and the ontology size and constructs. Further investigations are left to more extensive experimental campaigns.

Based on the case study as well as on early experiences by thesis students and interns of the research group laboratory, using EVOWLUATOR to perform tests on real-world reasoners appears to be generally straightforward, while still providing a good degree of flexibility. The integration of six desktop reasoners and five

Table 12
BioPortal — Correlation between time, memory peak and energy footprint score on Linux.

Reasoner	Energy–Time	Energy–Memory	Time–Memory
Fact++	0.61	0.70	0.42
HermiT	0.69	0.53	0.36
Konclude	0.89	0.91	0.97
TrOWL	0.97	0.97	0.92

Table 13
BioPortal — Correlation between time, memory peak and energy footprint score on macOS.

Reasoner	Energy–Time	Energy–Memory	Time–Memory
Fact++	0.61	0.63	0.37
HermiT	0.70	0.63	0.69
Konclude	0.99	0.98	0.96
TrOWL	0.98	0.98	0.98

Table 14
BioPortal — Correlation between energy footprint scores on Linux and macOS.

Reasoner	Energy Correlation
Fact++	0.97
HermiT	0.99
Konclude	0.90
TrOWL	0.97

additional configurations for mobile tests has required only about 250 lines of Python code. The visualization functionality has come in particularly handy, as the summaries and plots it provides allow for a quick at-a-glance performance comparison.

4. OWL reasoner evaluation framework

EVOWLUATOR has been designed with flexibility in mind, particularly concerning the ability to test multiple reasoning engines and the capability to run inference services on mobile and embedded devices.

To achieve these goals, EVOWLUATOR follows the *object-oriented* paradigm: user configuration involves extending Python abstract base classes in the framework with concrete subclasses implementing their parents' interfaces. Compared to a *declarative* solution, *e.g.*, structured configuration files, this *programmatic* approach may be more verbose, as it requires the implementation of small code modules, but it is much more expressive, as it enables the usage of the Python programming language along with its standard library. This in turn makes the technique flexible enough to allow the integration of arbitrary reasoner interfaces.

The following subsections outline architecture and main features of the proposed framework.

4.1. Architecture

EVOWLUATOR runs on GNU/Linux, macOS, and Windows (through the *Windows Subsystem for Linux*, WSL). Fig. 11 depicts a high-level architecture and data flow overview. The framework currently allows the assessment of the *ontology classification* and *consistency* standard reasoning tasks, as well as the *matchmaking* non-standard inference recalled in Section 2. As anticipated in Section 3.3, they can be tested under three *evaluation modes*:

- **Correctness:** validation of inference results by using a reference reasoner as test oracle.
- **Performance**: assessment of turnaround time for ontology loading/parsing and reasoning, and peak memory usage as the maximum resident set size of the reasoner process.
- **Energy footprint**: estimation of the energy employment for the reasoner.

Main components of the framework are reported hereafter and pictured in the UML component diagram in Fig. 12. The *Data* module exposes the `Dataset` and `Ontology` classes, which allow access to user-provided datasets and ontologies contained therein. The *Evaluation engine* interfaces with reasoners, ontologies and energy profilers via user-configurable software endpoints. Most of the functionality is provided by *Reasoning Evaluators*, subclasses of the `Evaluator` abstract class which implement business logic for each assessment task and mode (*e.g.*, ontology reasoning correctness, matchmaking performance, and so on). `InfoEvaluator` is a special class displaying information about reasoners and datasets. `Task` is the core API for spawning processes and capturing their output: its subclasses allow profiling energy consumption and benchmarking execution times and memory usage.

The *Evaluation engine* component invokes inference tasks implemented by reasoners through user-provided subclasses of the `Reasoner` abstract class, which supports running tests on the local machine as well as orchestrating them on remote devices. The latter option is mainly meant for mobile and embedded devices, with iOS and Android supported out-of-the-box (see Section 4.2). Performance and energy footprint tests can be run repeatedly for a number of *iterations* set via command line arguments. If more than one iteration is requested, the framework averages results before feeding them to subsequent steps of the processing pipeline. It is also possible to set a *timeout* for each reasoning task, after which the reasoner process is killed. Besides timeouts, EVOWLUATOR can detect *runtime errors* in two ways: the reasoner ends with a non-zero exit code, or it fails to output some mandatory data, *e.g.*, computation time for performance tests, or inference results for correctness tests. Once a test is completed, the *Evaluation engine* outputs and stores the following items:

- A human-readable *log* of the assessment;
- Machine-processable test *outcomes*. Both contents and structure may vary according to evaluation task and mode.
- *Configuration* used for the evaluation (selected reasoners, dataset, syntaxes, etc.).

The latter two items are used as inputs to the *Visualization engine*, which generates tabular summaries and graphical views of the evaluation outcomes, providing a human-understandable performance recap. This component is implemented by means of subclasses of the `Visualizer` abstract class. Plotting functionality is provided by subclasses of the `Plot` abstract class. More details on visualization are in Section 4.4.

4.2. Reasoners interface

In order to be invoked by EVOWLUATOR locally, reasoners must have a command line interface able to run reasoning tasks on specific ontologies: basically, they must at least accept an indication of the reasoning task to carry out, and the path to an ontology file. The exact structure of the arguments is specified on a per-reasoner basis by implementing the `Reasoner` interface, as described below. With regards to expected outputs, reasoners must print parsing and reasoning time separately to the standard output, which allows for the comparison of turnaround times. Concerning the evaluation of correctness, after classification the reasoner must output the inferred taxonomy to a file in an OWL serialization syntax supported by the OWL API [27]. Consistency results can be simply printed to standard output, instead. The expected output format is detailed in the documentation.[18]

[18] EVOWLUATOR documentation: http://swot.sisinflab.poliba.it/evowluator/.

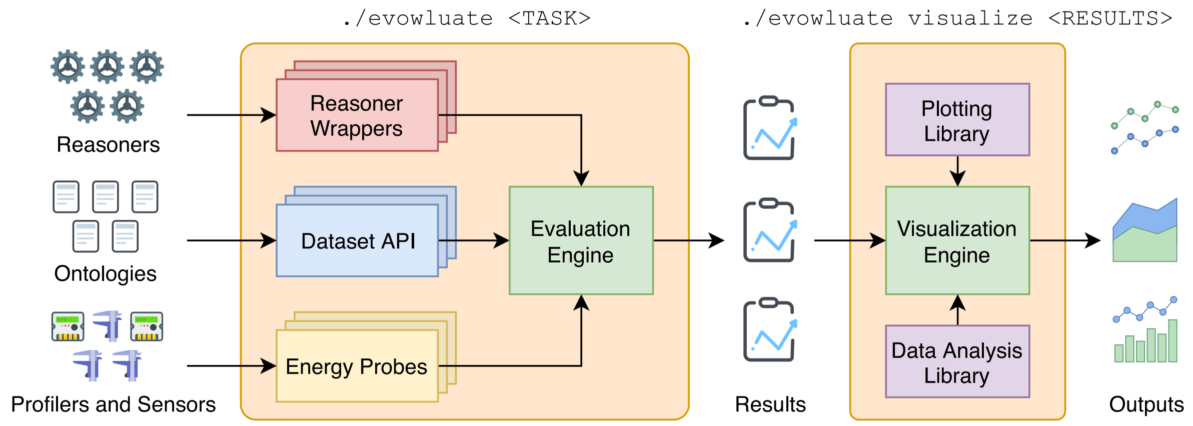


Fig. 11. High-level architecture and data flow.

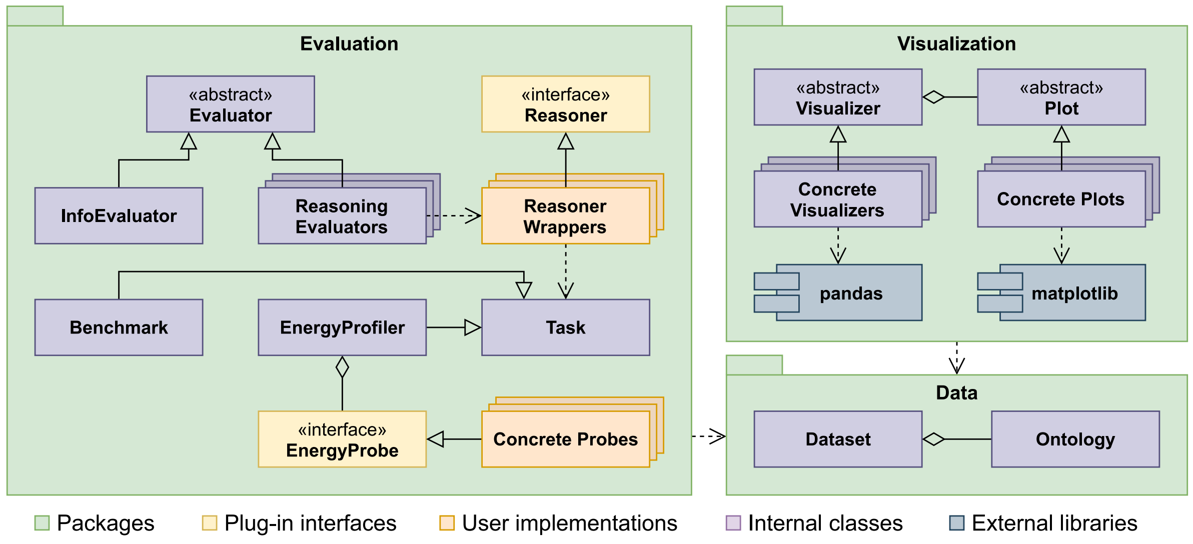


Fig. 12. UML diagram of the main components of the framework.

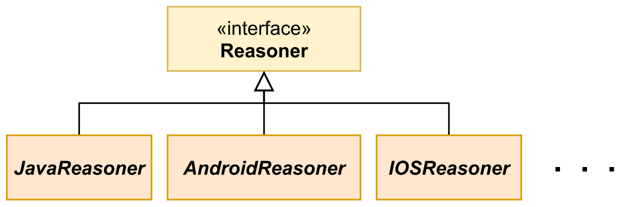


Fig. 13. *Reasoner* interface and template classes.

The framework adopts a plug-in architecture to enable the expansion of reasoner support. Inference engines complying with the above prerequisites can be configured by subclassing the provided `Reasoner` abstract base class, which allows them to be loaded dynamically. Besides metadata for the wrapped reasoner, such as name, path to the executable file, supported OWL syntaxes and inference services, the subclass must specify the command line argument array for each supported reasoning task. Concerning correctness results, they can either be provided according to the output format expected by EVOWLUATOR (details are in the documentation), or it is possible to provide custom output parsing logic by subclassing the `ResultsParser` class.

In addition to the `Reasoner` base class, the proposed framework provides a few templates to simplify the integration of inference engines on notable platforms, pictured in Fig. 13 and described hereafter.

Java SE: this template facilitates the integration of Java reasoning engines compiled in *jar* files by abstracting away the instantiation of the *Java Virtual Machine* (JVM). JVM configuration is controlled through appropriate flags specified with dedicated methods.

iOS: the template enables running and testing iOS-based reasoners. In this case, inference engines have to be wrapped in *Xcode* projects, and specifically as Xcode test cases, *i.e.*, `XCTestCase` subclasses. Supported reasoning tasks are exposed by means of dedicated test case methods to be deployed to the target device, together with datasets for the evaluation. EVOWLUATOR invokes test cases through `xcodebuild`, Xcode's command line interface, passing any required data via environment variables. In this case, in the user-provided extension of the template, methods just need to return project-related information, such as the path to the Xcode project and the name of the test methods implementing each of the supported reasoning tasks.

Android: the template allows the framework to run and test reasoning tasks on Android devices. Similarly to iOS, user-implemented methods just need to return Android-specific information, such as the package identifier of the reasoner app.

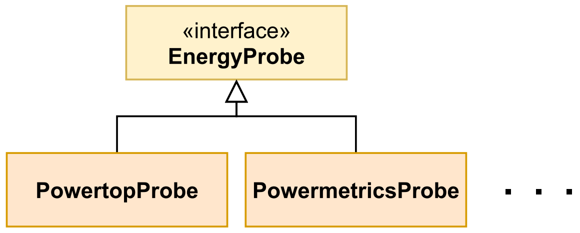


Fig. 14. *EnergyProbe* interface and built-in probes.

Users must install reasoners as Android applications containing an EVOWLUATE intent filter. EVOWLUATOR automatically installs a *launcher* application used to start reasoner apps by issuing appropriate EVOWLUATE intents, and to close them once the reasoning task is over. The implementation of this component exploits the Android Instrumentation class.[19] All communications between EVOWLUATOR and the launcher application are carried out through the *Android Debug Bridge* (adb).

4.3. Energy footprint

Energy drain estimation is implemented by the EnergyProfiler class, which runs the reasoner and polls a user-specified *energy probe* instance while the related process is alive. Energy probes must implement the EnergyProbe interface, as shown in Fig. 14. The *N* collected samples are then used to compute an *energy footprint* score:

$$score = sampling\_interval * \sum_{i=1}^{N} sample_i \qquad (1)$$

This provides an estimation of the energy employed by the engine during its execution.

The proposed approach aims at maximum compatibility with software- and hardware-based energy metering. Energy probes can either wrap power management tools integrated in the operating system, or get data from energy profilers like the ones surveyed in Section 5, or even capture readings from external hardware equipment, *e.g.*, the Monsoon High Voltage Power Monitor.[20] EVOWLUATOR comes with built-in support for PowerMetrics and PowerTOP software-based energy profilers, respectively developed by Apple Inc. for macOS and by Intel Corp. for GNU/Linux; the latter also works on Microsoft Windows through the WSL. Additional energy probes can be integrated by providing classes implementing the EnergyProbe interface, which must compute and store samples proportional to the average power usage during the time period between two consecutive polls.

Energy consumption is returned as an absolute *score* without units of measurement: this is required for compatibility with certain energy profilers, such as PowerMetrics, which do not output power usage in a physically relevant unit. However, by virtue of Eq. (1), if captured power samples are in watts, then the score can be interpreted as the energy usage in joules: this is the case of PowerTOP and, possibly, of hardware-based power meters.

For battery-equipped devices, care has to be taken to start each test with the same power source (grid or battery) and charge level, because they may influence values reported by probes, impairing test reproducibility. Finally, the framework should be used to carry out cross-device or cross-operating-system energy comparisons only when adopting an external hardware power meter, as it is the only reliable way to measure used energy in device-independent terms. Even in that case, analysis of the outcomes must carefully look for bias or confounding factors due to the inherent hardware/software differences among tested devices.

4.4. Results and report generation

Reports are generated by the *Visualization* component. Their contents vary based on reasoning task and evaluation mode. Raw data produced by the *Evaluation engine* are processed via the *pandas*[21] Python data analysis library. The framework computes aggregated results and outputs tables in Comma-Separated Value (CSV) format. Depending on the evaluation mode, EVOWLUATOR outputs the following items:

- **Correctness**: statistics about the number of correct and incorrect results, runtime errors and timeouts.
- **Performance**: per-ontology and dataset-wide parsing and reasoning times, as well as information about the minimum, maximum and average detected memory peak, for each reasoner.
- **Energy**: per-ontology and dataset-wide minimum, maximum and average energy scores, for each reasoner.

The framework also provides graphical views by exploiting the *Matplotlib*[22] library: plots are displayed in an interactive window, which can be used for navigation, zooming and cropping. The graphical interface also allows saving plots as vector or raster graphics files. Examples for that are in Section 3.4.

5. Related work

Approximately until 2010, reasoner evaluation has been dominated by benchmarks based on a relatively low number of ontologies and small sets of hand-crafted queries [28]. The *Lehigh University Benchmark (LUBM)* [29] is one of the most representative and popular specimens: it consists of one ontology on the domain of universities, fourteen extensional queries testing several properties, and a synthetic data generator to create scalable ABoxes. In [30] LUBM was used together with other three ontologies and a set of queries in four different OWL fragments, to compare five reasoners and identify the most adequate ones for each class of ontologies and corresponding inference tasks. Analogously, [31] compared eight reasoners on classification, consistency, three concept satisfiability queries and four subsumption checks, using three large ontologies in the OWL 2 EL profile. In [32] LUBM was extended to support SPARQL-based stream reasoning.

Thanks to the expanding availability of real-world knowledge bases and knowledge graphs for various application domains, in later years large and diverse corpuses have been created for OWL reasoner benchmarking, in order to evaluate systems under realistic conditions. In 2012 the classification time of 4 reasoners was measured on a dataset of over 300 real-world ontologies, a record at the time of publication [33]; as a further peculiarity, ontology metrics were used as features in a machine learning problem to predict processing time. The *OWL Reasoner Evaluation* (ORE) workshop series hosted a yearly competition from 2012 to 2016, with growing numbers of ontologies (in various OWL

[19] Instrumentation: https://developer.android.com/reference/android/app/Instrumentation.

[20] Monsoon High Voltage Power Monitor: https://www.msoon.com/high-voltage-power-monitor.

[21] Pandas home: https://pandas.pydata.org.

[22] Matplotlib home: https://matplotlib.org.

profiles), reasoning tasks and competing systems [24,34]. Score was determined by the number of problems solved out of the total in each competition track; time was used to break ties for the final standings. Unfortunately, other performance indexes like memory or energy usage were not considered. The growth of datasets and test cases have evidenced the need for tools able to automate the benchmarking: the framework developed in [24] is still to date one of the most scalable and versatile solutions to do that on traditional computer platforms. Adapting that system to support additional metrics and novel target platforms was appraised before starting the evOWLuator project, though the required effort was assessed as too high, as the framework is mainly oriented to "live" reasoner competitions and focuses on the correctness of inferences. Extending the ORE framework would have implied adding required features like memory and energy evaluation or mobile platforms support to an already large software project (over 200 Java source files and more than 15000 lines of code overall, without accounting for the Web interface) which was not designed specifically for expandability and included unnecessary features for the envisioned use cases. In comparison, evOWLuator has a compact codebase (26 Python source files, fewer than 3000 lines of code) and is customizable via plugins with no modifications to the existing source code.

Evaluation frameworks specialized for non-standard inference test cases also exist. *JustBench* [17] analyzes reasoner performance on testing the correctness of *justifications*, *i.e.*, minimal ontology subsets for an entailment to hold. Experiments concerning the *Mini-ME* (Mini Matchmaking Engine) Java/Android reasoner [14] and its *Swift* reengineering for iOS [13] have evaluated performance of the non-standard Concept Abduction/Contraction *matchmaking* task, recalled in Section 2. More recently, the interest in applying semantic technologies to ubiquitous computing has generated the need for benchmarking mobile-oriented OWL profiles and emerging mobile reasoners. The experimental campaign in [12] has evaluated six reasoners with *OWL API* [27] support on Android, using the ORE 2013 dataset [34], on ontology classification and consistency tasks. In order to automate the large number of tests, an Android application has been developed: it allows selecting the reasoner, the set of ontologies (based on an OWL profile sublanguage) and the inference task, and then saves results in an embedded database.

Platform heterogeneity and strict energy usage control are among the distinctive traits of mobile and ubiquitous computing. Therefore cross-platform and energy-aware benchmarking frameworks are currently at the edge of research and development efforts. The framework in [35], aimed at evaluating mobile semantic rule engines, has been developed in JavaScript exploiting the *PhoneGap*[23] Software Development Kit (SDK): this approach allows harnessing rule engines written either in JavaScript or natively for one of the platforms supported by PhoneGap (Android, iOS, Windows 8.1). A recent and enhanced version of the framework, named *MobiBench* [36], additionally supports OWL 2 RL reasoning, benchmark automation, and Java reasoners via the *Nashorn* JavaScript engine included in Java Standard Edition (SE) version 8 and later. Energy usage profiling is planned for future work.

A few energy-aware mobile benchmarks exist, but they mostly appear as one-off efforts for specific investigations. Conversely, the proposal released in this paper is aimed at the whole research community for usage in a range of experimental settings and use cases. The energy benchmark in [25] analyzes Android reasoners by replacing the testbed device battery with a hardware power monitor, connected to a laptop for data capture. While this method guarantees accuracy, setup is very complicated, as electrical parameters are different for each mobile device model. Furthermore, recent design trends have been increasingly adopting non-removable batteries, making the tests even more difficult. An *ODROID XU3* single-board computer – which integrates power monitoring circuitry – is used in [37] to evaluate energy usage of six reasoners. Even though more practical than [25], the adopted hardware does not match real-world mobile and ubiquitous scenarios; furthermore, the study only includes Java-based reasoners. Conversely, the purely software-based energy profiler in [38] works by correlating battery state of charge and per-method time measurements, both collected via Android APIs available on most recent devices. It exhibits accuracy within 5% of hardware-based monitoring for apps that do not make heavy usage of network connections or embedded sensors (*i.e.*, the typical situation of profiling OWL reasoning tasks). A similar approach is adopted in [39] to benchmark energy consumption of Android reasoners. The latter tool is used also in [26] to derive a prediction model of energy consumption for ontology reasoning over mobile devices. Experiments yield two particularly interesting results: (i) given a device and a reasoning task, energy consumption is influenced by the battery charge state; (ii) not all reasoner/device pairs exhibit a linear correlation between time and power consumption, *i.e.*, the longest tasks are not always the most energy-intensive. These outcomes call for further investigations: an evaluation framework allowing tests to scale efficiently to a range of mobile devices and platforms can provide insights.

6. Conclusion

This paper has proposed a novel multi-platform and energy-aware framework for OWL reasoner benchmarking. The first release allows evaluating correctness, performance (time and memory peak) and energy footprint of a set of standard and non-standard reasoning tasks. Support for both desktop and mobile platforms, scalability and flexibility are some of the most relevant core features. Automatic report generation in tabular and plot forms facilitates the presentation of experimental outcomes.

Future work on the framework will expand the set of supported reasoning services. Further planned improvements include: native Windows compatibility; reliable energy probe implementations for more platforms (starting with Android and iOS); built-in support for selected embedded platforms; more visualization types. Finally, a systematic survey of all actively developed reasoning engines for desktop and mobile systems, including energy footprint evaluation, is planned as a primary research interest.

In addition to long-term project maintenance and development, fostering the adoption of the framework in the Semantic Web community is a crucial goal. To this end, a range of activities is about to start, including: active management of the user community; submission of proposals for demos, tutorials, and challenges to major Semantic Web venues; exploitation in collaborative project on the evaluation of OWL reasoning from specific perspectives and/or in particular applications. Community adoption will, in turn, provide useful feedback for improving the framework and establishing priorities for new features and enhancements.

CRediT authorship contribution statement

Floriano Scioscia: Conception and design of study, Acquisition of data, Analysis and/or interpretation of data, Writing – original draft, Writing – review & editing. **Ivano Bilenchi:** Conception and design of study, Acquisition of data, Analysis and/or interpretation of data, Writing – original draft, Writing – review & editing.

[23] PhoneGap home: https://phonegap.com/. It is based on the open source *Apache Cordova* engine: https://cordova.apache.org/.

Michele Ruta: Conception and design of study, Acquisition of data, Analysis and/or interpretation of data, Writing – original draft, Writing – review & editing. **Filippo Gramegna:** Conception and design of study, Acquisition of data, Analysis and/or interpretation of data, Writing – original draft, Writing – review & editing. **Davide Loconte:** Conception and design of study, Acquisition of data, Analysis and/or interpretation of data, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

All authors approved the version of the manuscript to be published.

Funding

This work has been supported by Italian Ministry of Economic Development R&D project *BARIUM5G* (Blockchain and ARtificial Intelligence for Ubiquitous coMputing via 5G) and by Italian PON project *RPASInAir* (Remotely Piloted Aircraft Systems Integration in non-segregated Air space for services).

Intellectual property

We confirm that we have given due consideration to the protection of intellectual property associated with this work and that there are no impediments to publication, including the timing of publication, with respect to intellectual property. In so doing we confirm that we have followed the regulations of our institutions concerning intellectual property.

References

[1] B. Parsia, S. Rudolph, M. Krötzsch, P. Patel-Schneider, P. Hitzler, OWL 2 Web Ontology Language Primer, W3C Recommendation, second ed., W3C, 2012, http://www.w3.org/TR/owl2-primer.
[2] The W3C SPARQL Working Group, SPARQL 1.1 Overview, W3C Recommendation, W3C, 2013, https://www.w3.org/TR/sparql11-overview/.
[3] N. Matentzoglu, J. Leo, V. Hudhra, U. Sattler, B. Parsia, A survey of current, stand-alone OWL reasoners, in: 4th OWL Reasoner Evaluation Workshop, ORE, in: CEUR Workshop Proceedings, CEUR-WS, Aachen, Germany, vol. 1387, 2015, pp. 68–79.
[4] A. Zaidan, B. Zaidan, M. Hussain, A. Haiqi, M.M. Kiah, M. Abdulnabi, Multi-criteria analysis for OS-EMR software selection problem: A comparative study, Decis. Support Syst. 78 (2015) 15–27.
[5] F. Scioscia, M. Ruta, Building a Semantic Web of Things: issues and perspectives in information compression, in: Proceedings of the 3rd IEEE International Conference on Semantic Computing, IEEE Computer Society, Piscataway, NJ, USA, 2009, pp. 589–594.
[6] D. Pfisterer, K. Romer, D. Bimschas, O. Kleine, R. Mietz, C. Truong, H. Hasemann, A. Kroller, M. Pagel, M. Hauswirth, et al., SPITFIRE: Toward a Semantic Web of Things, IEEE Commun. Mag. 49 (11) (2011) 40–48.
[7] A.J. Jara, A.C. Olivieri, Y. Bocchi, M. Jung, W. Kastner, A.F. Skarmeta, Semantic Web of Things: an analysis of the application semantics for the IoT moving towards the IoT convergence, Int. J. Web Grid Serv. 10 (2–3) (2014) 244–272.
[8] M. Noura, A. Gyrard, S. Heil, M. Gaedke, Automatic knowledge extraction to build Semantic Web of Things applications, IEEE Internet Things J. 6 (5) (2019) 8447–8454.
[9] N. Abbas, Y. Zhang, A. Taherkordi, T. Skeie, Mobile Edge Computing: A survey, IEEE Internet Things J. 5 (1) (2017) 450–465.
[10] S. Grimm, M. Watzke, T. Hubauer, F. Cescolini, Embedded $\mathcal{EL}$+ reasoning on programmable logic controllers, in: International Semantic Web Conference, Springer, 2012, pp. 66–81.
[11] M. Bermúdez-Edo, E. Della Valle, T. Palpanas, Semantic challenges for the variety and velocity dimensions of Big Data, Int. J. Semant. Web Inf. Syst. 12 (2016).
[12] C. Bobed, R. Yus, F. Bobillo, E. Mena, Semantic reasoning on mobile devices: Do androids dream of efficient reasoners? J. Web Semant. 35 (2015) 167–183.
[13] M. Ruta, F. Scioscia, F. Gramegna, I. Bilenchi, E. Di Sciascio, Mini-ME Swift: the first OWL reasoner for iOS, in: 16th Extended Semantic Web Conference, ESWC 2019, in: Lecture Notes in Computer Science, no. 11503, Springer, 2019, pp. 298–313.
[14] F. Scioscia, M. Ruta, G. Loseto, F. Gramegna, S. Ieva, A. Pinto, E. Di Sciascio, Mini-ME matchmaker and reasoner for the Semantic Web of Things, in: Innovations, Developments, and Applications of Semantic Web and Information Systems, IGI Global, Hershey, PA, USA, 2018, pp. 262–294.
[15] N. Guarino, D. Oberle, S. Staab, What is an ontology? in: Handbook on Ontologies, Springer, 2009, pp. 1–17.
[16] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, P. Patel-Schneider, The Description Logic Handbook, Cambridge University Press, 2002.
[17] S. Bail, B. Parsia, U. Sattler, Justbench: a framework for OWL benchmarking, in: International Semantic Web Conference, Springer, 2010, pp. 32–47.
[18] D. Tsarkov, I. Horrocks, FaCT++ Description Logic reasoner: System description, in: International Joint Conference on Automated Reasoning, IJCAR, Springer, Berlin, Germany, 2006, pp. 292–297.
[19] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, Z. Wang, HermiT: an OWL 2 reasoner, J. Automat. Reason. 53 (3) (2014) 245–269.
[20] A. Steigmiller, T. Liebig, B. Glimm, Konclude: system description, J. Web Semant. 27 (2014) 78–85.
[21] E. Thomas, J.Z. Pan, Y. Ren, TrOWL: Tractable OWL 2 reasoning infrastructure, in: Extended Semantic Web Conference, ESWC, Springer, Berlin, Germany, 2010, pp. 431–435.
[22] E. Sirin, B. Parsia, B.C. Grau, A. Kalyanpur, Y. Katz, Pellet: A practical OWL-DL reasoner, J. Web Semant. 5 (2) (2007) 51–53.
[23] P.L. Whetzel, N.F. Noy, N.H. Shah, P.R. Alexander, C. Nyulas, T. Tudorache, M.A. Musen, Bioportal: enhanced functionality via new web services from the National Center for Biomedical Ontology to access and use ontologies in software applications, Nucleic Acids Res. 39 (suppl_2) (2011) W541–W545.
[24] B. Parsia, N. Matentzoglu, R.S. Gonçalves, B. Glimm, A. Steigmiller, The OWL reasoner evaluation (ORE) 2015 competition report, J. Automat. Reason. 59 (4) (2017) 455–482.
[25] E.W. Patton, D.L. McGuinness, A power consumption benchmark for reasoners on mobile devices, in: International Semantic Web Conference, Springer, 2014, pp. 409–424.
[26] I. Guclu, Y.-F. Li, J.Z. Pan, M.J. Kollingbaum, Predicting energy consumption of ontology reasoning over mobile devices, in: International Semantic Web Conference, Springer, 2016, pp. 289–304.
[27] M. Horridge, S. Bechhofer, The OWL API: A Java API for OWL ontologies, Semant. Web 2 (1) (2011) 11–21.
[28] M. Imprialou, G. Stoilos, B.C. Grau, Benchmarking ontology-based query rewriting systems, in: Twenty-Sixth AAAI Conference on Artificial Intelligence, 2012, pp. 779–785.
[29] Y. Guo, Z. Pan, J. Heflin, LUBM: A Benchmark for OWL knowledge base systems, Web Semant.: Sci., Serv. Agents World Wide Web 3 (2–3) (2005) 158–182.
[30] J. Bock, P. Haase, Q. Ji, R. Volz, Benchmarking OWL Reasoners, in: ARea2008 – Workshop on Advancing Reasoning on the Web: Scalability and Commonsense, 2008, pp. 1.1–1.15.
[31] K. Dentler, R. Cornet, A. Ten Teije, N. De Keizer, Comparison of reasoners for large ontologies in the OWL 2 EL profile, Semant. Web 2 (2) (2011) 71–87.
[32] T.N. Nguyen, W. Siberski, SLUBM: An extended LUBM benchmark for stream reasoning, in: 2nd International Workshop on Ordering and Reasoning, in the 12th International Semantic Web Conference, ISWC 2013, in: CEUR Workshop Proceedings, vol. 1059, 2013.
[33] Y.-B. Kang, Y.-F. Li, S. Krishnaswamy, A rigorous characterization of classification performance – a tale of four reasoners, in: I. Horrocks, M. Yatskevich, E. Jimenez-Ruiz (Eds.), OWL Reasoner Evaluation Workshop, ORE 2012, in: CEUR Workshop Proceedings, CEUR-WS, vol. 858, 2012, pp. 88–99.
[34] R.S. Gonçalves, S. Bail, E. Jiménez-Ruiz, N. Matentzoglu, B. Parsia, B. Glimm, Y. Kazakov, OWL reasoner evaluation (ORE) workshop 2013 results, in: ORE, 2013, 1–18.
[35] W. Van Woensel, N. Al Haider, A. Ahmad, S.S. Abidi, A cross-platform benchmark framework for mobile Semantic Web reasoning engines, in: International Semantic Web Conference, Springer, 2014, pp. 389–408.

[36] W. Van Woensel, S.S.R. Abidi, Benchmarking semantic reasoning on mobile platforms: Towards optimization using OWL2 RL, Semant. Web 10 (4) (2019) 637–663.
[37] P. Koopmann, M. Hähnel, A.-Y. Turhan, Energy-efficiency of OWL reasoners – frequency matters, in: Joint International Semantic Technology Conference, Springer, 2017, pp. 86–101.
[38] D. Di Nucci, F. Palomba, A. Prota, A. Panichella, A. Zaidman, A. De Lucia, Software-based energy profiling of Android apps: Simple, efficient and reliable? in: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering, SANER, IEEE, 2017, pp. 103–114.
[39] E. Valincius, H.H. Nguyen, J.Z. Pan, A power consumption benchmark framework for ontology reasoning on Android devices, in: OWL Reasoner Evaluation (ORE) Workshop, 2015, 80–86.