Ruslan Sakevych
github.com/lionell

rsakevych@gmail.com
1-650-391-5608

Education

- **Taras Shevchenko National University** — Kyiv, Ukraine
  M.S. in Computer Science; GPA: 4.92/5.0 — *Sep 2018 – Jun 2020*

- **Taras Shevchenko National University** — Kyiv, Ukraine
  B.S. in Computer Science; GPA: 4.97/5.0 — *Sep 2014 – Jun 2018*

Experience

- **Facebook** — Seattle, WA
  Software Engineer — *Oct 2019 - Present*
  - Designed and implemented service that computes Probabilistic Flakiness Score in real-time via Bayesian inference.
  - Scaled PFS service CPU utilization 10x, latency 20x, number of replicas 7.5x via profiling/caching/parallelism.
  - Post with details: https://engineering.fb.com/2020/12/10/developer-tools/probabilistic-flakiness/
  - Replaced statistical model with stochastic ML one for sizing(memory-wise) CI workflows.
  - New Job Sizer model reduced memory utilization by 30% and out-of-memory rate by 17%.
  - Achieved another 12% improvement in MSLE by doing feature engineering for Job Sizer model.
  - Prototyped an ML based model for the CI workflow ETA estimation.
  - It reduced MSLE by 50% and allows to control confidence levels of the prediction (thx to Monte-Carlo dropout).
  - This gives a way to show uncertanty to user and not only point-estimate (UX research finds this misleading).

- **Google** — Sunnyvale, CA
  Software Engineering Intern — *May 2019 - Aug 2019*
  - Designed and implemented an automatic build memory regression finder that operated at Google scale.
  - Created a developer dashboard for troubleshooting memory regression issues.
  - Was able to automatically(via dashboard) pinpoint culprit changes for recent major regressions.

- **Google** — Sunnyvale, CA
  Software Engineering Intern — *Aug 2018 - Nov 2018*
  - Migrated old ML pipeline onto Tensorflow-backed framework TFX. Experience with data processing pipelines.
  - Implemented parallel n-ary search algorithm. Speeded up culprit finder 16x times (on millions of changes).
  - Designed more sophisticated and robust parallel batching algorithm. Reduced tail request latency 3x times.
  - Mined build graph of the whole Google using MapReduce. Did an attack on dependency set similarity problem.

- **Facebook** — London, UK
  Software Engineer Intern — *Jan 2018 - Mar 2018*
  - Rearchitected Hack parser to be reactive, allowing parsing to be inlined with the computation of the result.
  - 25% parse time reduction for the Hack type-checker (using most of the file contents) on the full-codebase.
  - Up to 50% speed up for tools that use less information(facts extraction) on hundreds of thousands of files.
  - Developed a toolset to analyze and remove unnecessary build dependencies, resulting in 2x speed up.

- **Microsoft** — Redmond, WA
  Software Developer Intern — *Jul 2017 - Oct 2017*
  - Engineered a new workflow to automate raw telemetry data aggregation and transformation.
  - System monitors execution of user-defined query and publishes results back to data warehouse.
  - Used for intermediate metrics aggregation to reduce data volumes and speed up queries.

- **Google** Sunnyvale, CA
  Software Engineering Intern *Apr 2017 - Jul 2017*
  - Research on build/test time prediction. Performed data analysis, model evaluation and feature engineering.
  - Created tools for ML models debugging/visualization and core service efficiency evaluation.
  - Investigated and mitigated incidents in complex build infrastructure at Google scale.

- **Google** Mountain View, CA
  Software Engineering Intern *May 2016 - Aug 2016*
  - Engineered a service that clasterizes build targets to reduce overall resources usage.
  - Performed evaluation of different batching strategies: memory, run-time optimization.
  - Trained ML models to predict build memory usage and avoid out of memory errors.

Projects

- **PARCS**: Communication Sequential Processes (CSP) like approach for language agnostic distributed computing.
- **PARCS autodiscovery**: LAN service autodiscovery for nodes in PARCS cluster based on UDP broadcasting.
- **Smart Pacmans**: Visualization of how neural networks can be trained using genetic algorithms. https://lionell.github.io/smart-pacmans
- **Resolution Theorem Prover**: Based on sequential method and operates in classic first-order logic.
- **Pollard-Rho**: Parallel implementation of Pollard-Rho algorithm in Go. **Extra**: Ethereum smart-contract impl.
- **Huffman+RLE**: Small (1% on average) optimization for Huffman algorithm followed by run-length-encoding.
- **Parallel PageRank**: Based on MPI and OpenMP. Ultimate goal was to compute Wikipedia pagerank.
- **Aqua Lang**: Data processing language that uses concepts from relational algebra. Opposite to declarative SQL.