Comprehensive Latency issue Diagnosis in Microservices Using Enhanced Spectrum Analysis

Maryam Ekhlasi

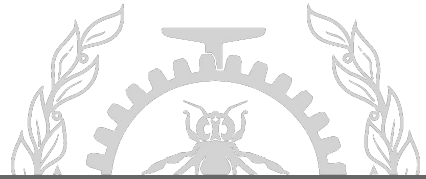
May 12th, 2025

Polytechnique Montreal

DORSAL Laboratory

Motivations

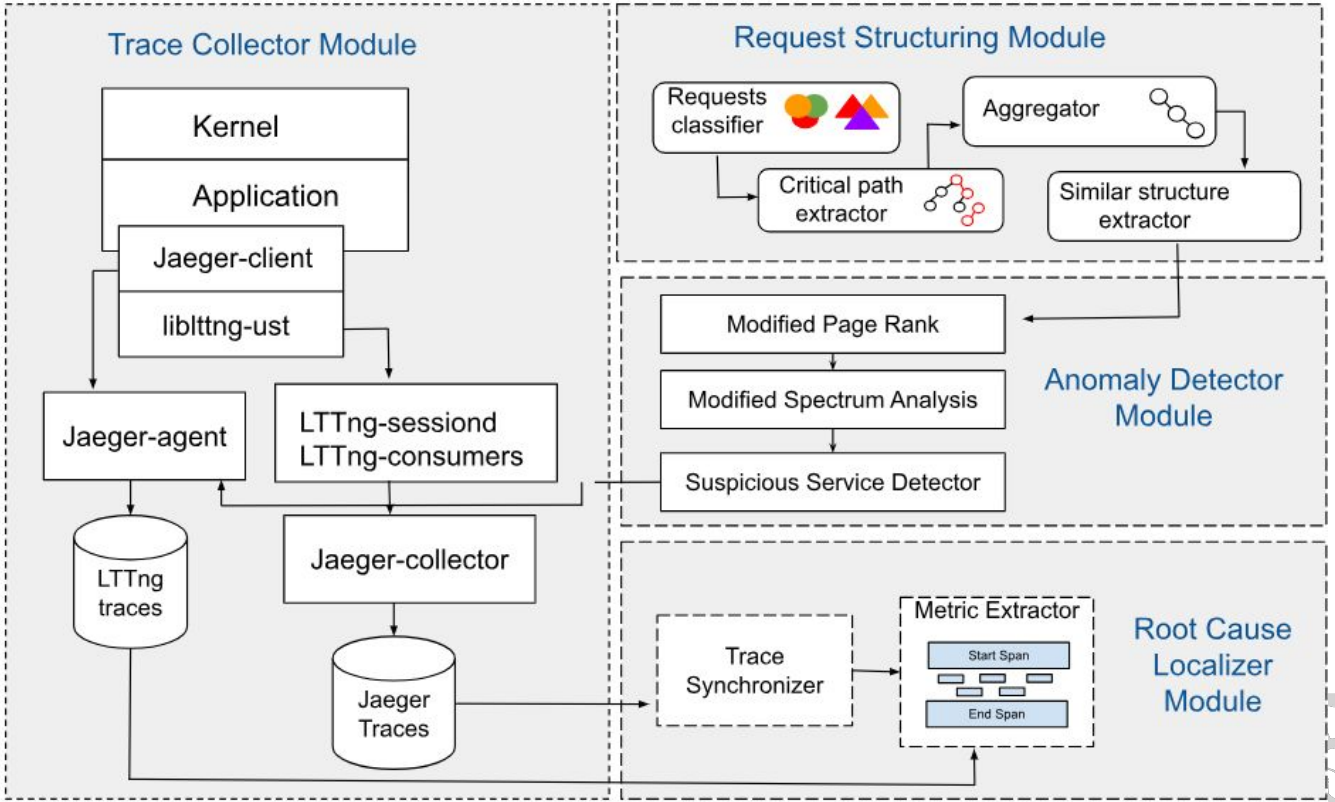
- **Manual trace analysis is time-consuming and requires expert knowledge**
  Investigating performance issues often involves deep domain expertise and hours of engineering effort.
- **Distributed traces are large and storage-intensive**
  High-volume microservices generate largenumbers of spans, consuming significant storage resources.
- **Traditional RCA approaches collect excessive and often irrelevant data**
  Uniform instrumentation leads to data overload, making it harder to isolate meaningful signals.
- **Performance slowdowns degrade user experience and impact revenue**
  Latency issues directly affect product responsiveness, customer satisfaction, and business outcomes.



Our Goal

- **Automate root cause detection in distributed microservices**
  - Identify the true source of performance issues without manual deep dives.
- **Minimize diagnostic overhead in production environments**
  - Collect detailed metrics only for suspicious spans, reducing storage and compute costs.
- **Improve accuracy in detecting critical services**
  - Focus analysis on the critical path and use ranking algorithms to highlight root causes.
- **Correlate service behavior with system-level performance data**
  - Combine application traces with kernel-level metrics (e.g., CPU, I/O).

Architecture



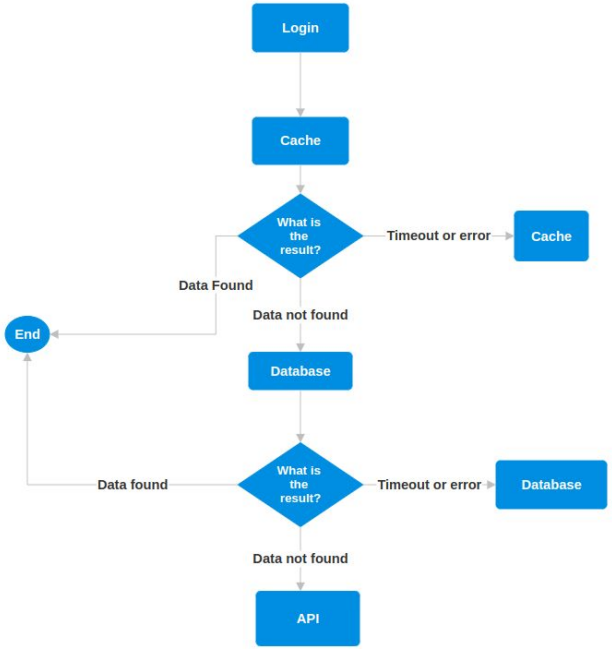
Real world Problem

Engineers want to understand occasional delays in their login processes. The login sequence typically follows these steps:

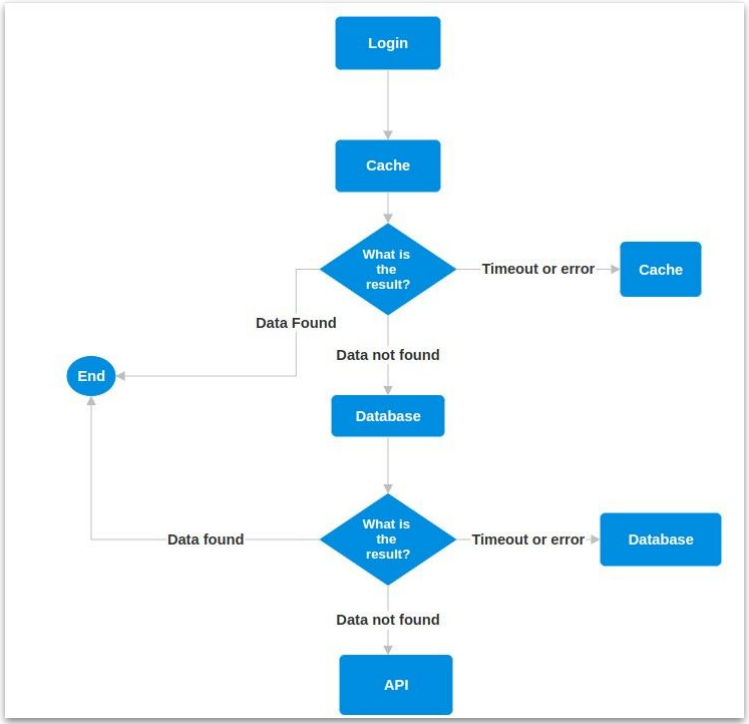
- Request information from the cache.
- Retry three times if the cache fails.
- Request data from the database if cache retrieval fails.
- Retry from an alternative database up to three times

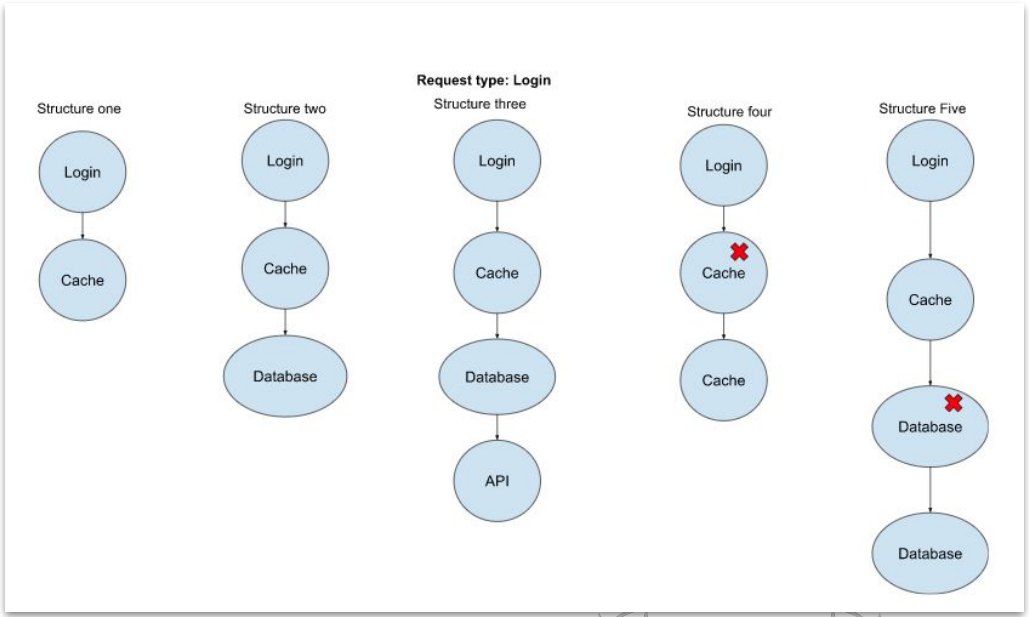
if the main database fails

- Query an external API if the secondary database fails.

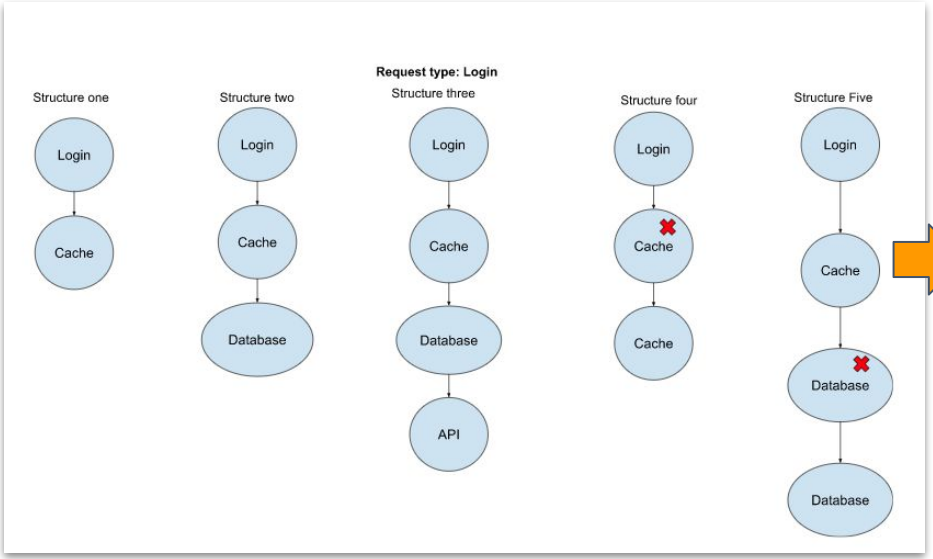


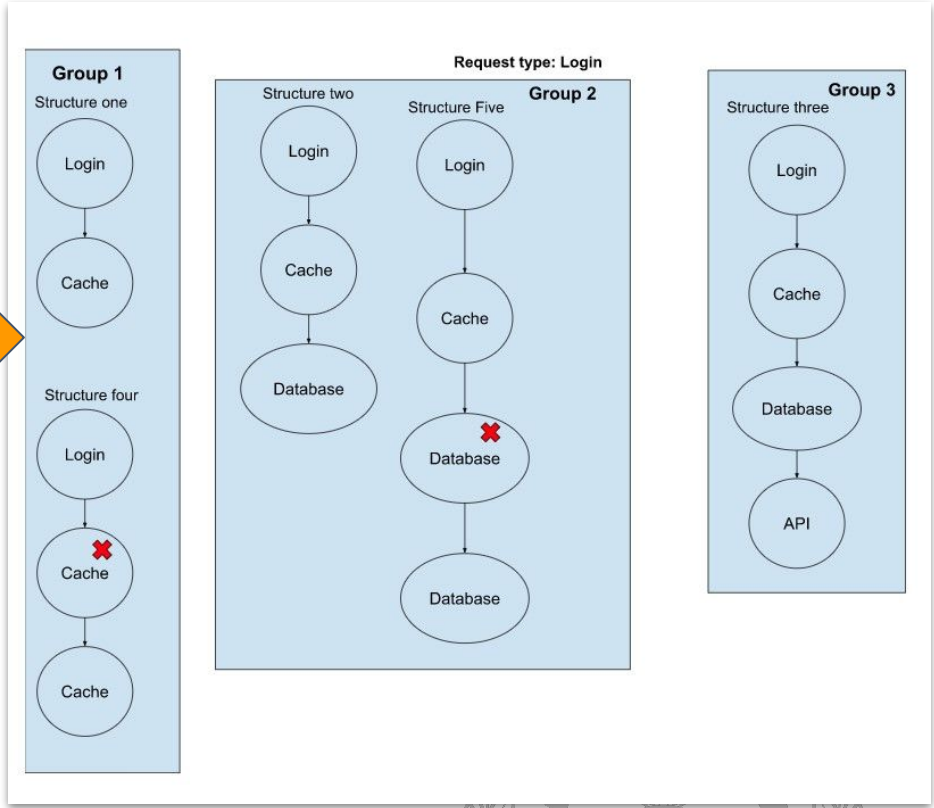
Example


Login
Cache
What is the result?
Timeout or error
Cache
Data Found
Data not found
End
Database
What is the result?
Data found
Timeout or error
Database
Data not found
API


Request type: Login
Structure one
Login
Cache
Structure two
Login
Cache
Database
Structure three
Login
Cache
Database
API
Structure four
Login
Cache
Cache
Structure Five
Login
Cache
Database
Database

Different Structures



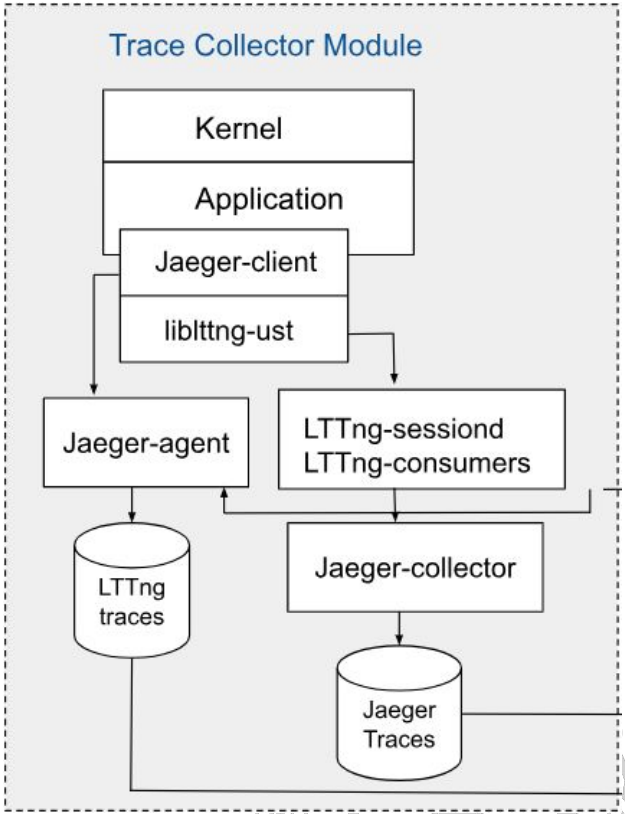


Concerns

- A single request may follow multiple structural patterns.
- Graph/tree mining approaches are often computationally expensive.
- Parallel service branches introduce challenges in trace analysis.
- Root cause analysis requires access to system-level information.
- Collecting system-level data can consume significant disk space.

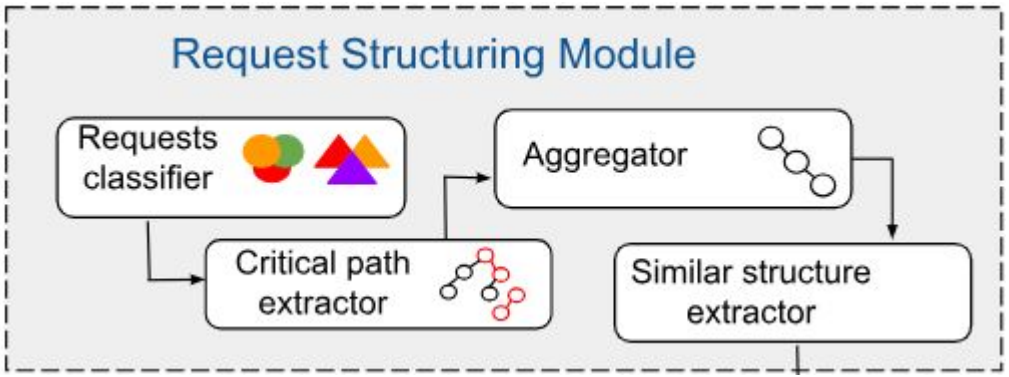
Strategy

- Two-level instrumentation of the application
- Collecting high-level traces



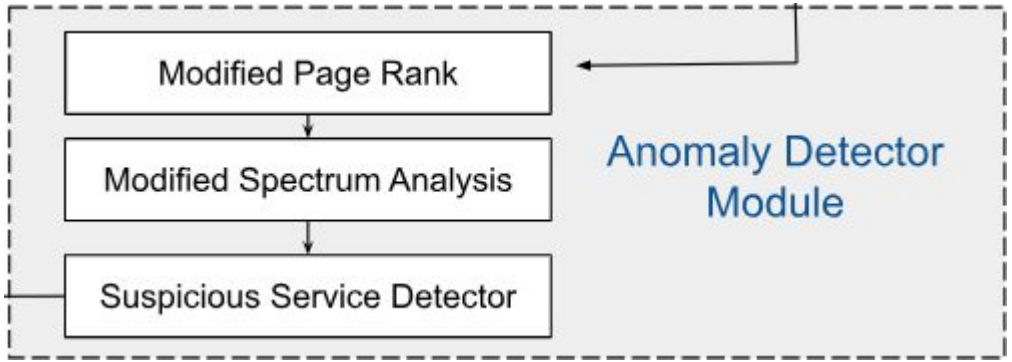
Strategy

- Grouping similar or identical traces for fair comparison and effective anomaly detection
- Extracting the critical path to isolate latency-contributing services and handle parallel calls
- Aggregating Identical service calls



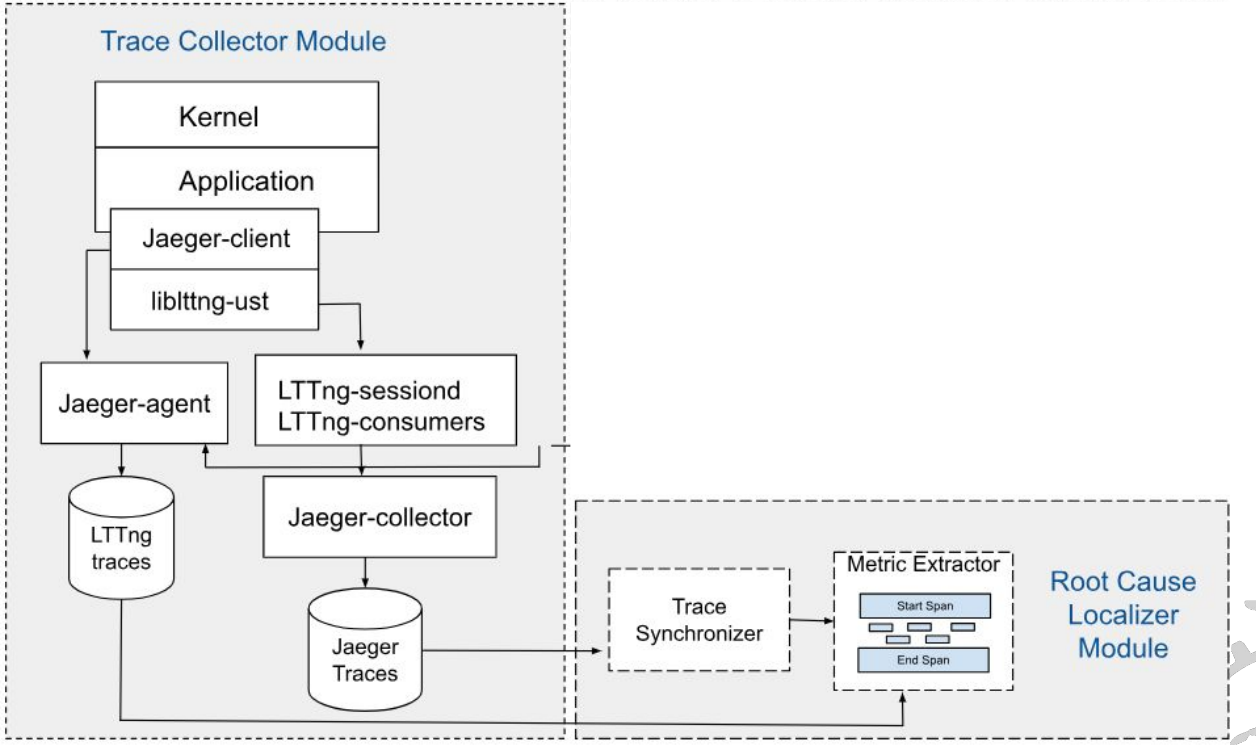
Strategy

- Evaluating Expected Latency per group
- Comparing the real latency and expected latency
- Personalised PageRank Algorithm
- Weighted Spectrum Ranker to list the order of suspicious services



Strategy

- Extract system calls for suspicious services for root cause analysis



Results

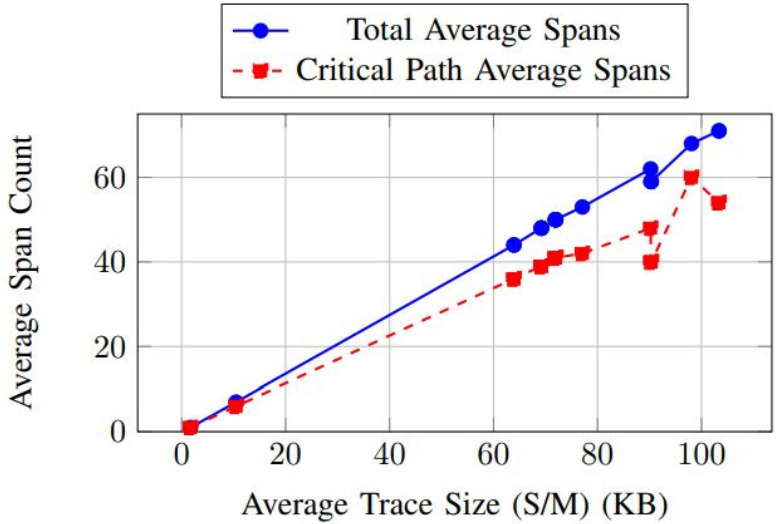
- Recall improved by 5.7% compared to state-of-the-art RCA tools (e.g., MicroRank, Nezha)
- Precision increased by 7% in identifying true root causes within microservice traces

Test Case	Recall (%)	Precision (%)	F1-score (%)
One RC HotRod	100	97	98
Two RC HotRod	100	94	97
One RC Train Ticket	97	91	94
Two RC Train Ticket	96	93	94
One RC Online Boutique	92	89	90
Two RC Online Boutique	90	85	87

Results

- Span volume reduced by 21% by focusing analysis on the critical path
- Storage usage dropped by over 99% through selective instrumentation
- Validated on multiple platforms: HotRod, TrainTicket, OnlineBoutique, and a real-world TiDB incident

Trace Size	Req.	Spans	UST	Kernel	Reduced Kernel	Kernel Size Reduction
29.6 MB	50	250	428 KB	29.2 MB	75 KB	99.74%
89.3 MB	250	1,250	1.3 MB	88.0 MB	375 KB	99.57%
326.4 MB	500	2,500	3.0 MB	323.3 MB	225 KB	99.93%
978.6 MB	3,000	150,000	17.5 MB	961 MB	300 KB	99.97%
1.7 GB	5,000	250,000	29.2 MB	1.7 GB	375 KB	99.98%



Thank you

Email: maryam.ekhlasi@polymtl.ca