



Thwarting Bobby Droptables: Adapting a TF-IDF HTTP Classifier to Embedded Hardware

September 2, 2009

Craig Ulmer
Maya Gokhale, Philip Top, John May

SNL/CA
LLNL

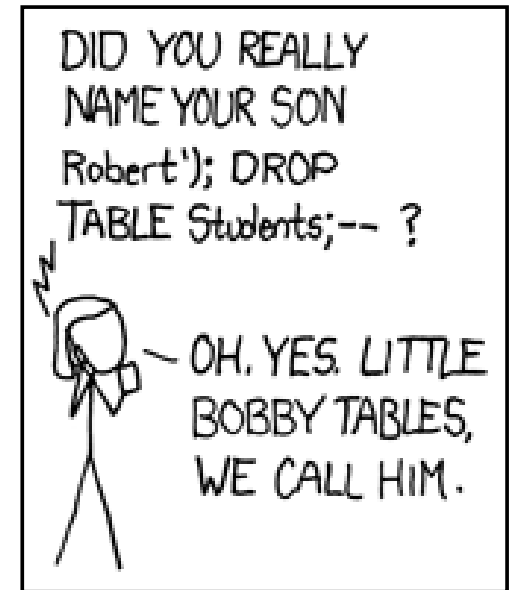


Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.



Network Security is Challenging

- Mixed requirements in server security
 - Provide flexible web services
 - Do so on top of existing standards
 - Thwart malicious behavior
- HTTP is conduit for web traffic
 - Simple, plain-text formatting
 - Gateway to databases, files, executables
- Malicious users also use these interfaces
 - Query a DB, invoke commands
 - Obfuscate commands, game network filters
- Can we embed more intelligent filtering in the network?



From xkcd, Randall Munroe
<http://xkcd.com/327>



Overview

- LLNL work on HTTP attack classification
- Can this be converted to run on embedded network hardware?
 - Tileria or FPGAs: limited memory, operate in streaming manner
 - Need to convert 160MB dictionary to ~128 KB
- Our approach
 - Hypothesis: number of terms more useful than exact probabilities
 - Compress dataset via truncation and hashing tricks
- Status
 - Implemented core hardware design for FPGAs
 - Porting C version to Tileria

ECML/PKDD 2007 Discovery Challenge

- HTTP Traffic Classification
 - Apply machine learning to identify malicious activity in HTTP
- Hand-labeled datasets of HTTP flows
 - Training: 50K inputs, 30% attacks
 - Competition: 70K inputs, 40% attacks
 - 7 Attack Types: XSS, SQL/LDAP/XPATH injection, path traversal, command execution, and SSI

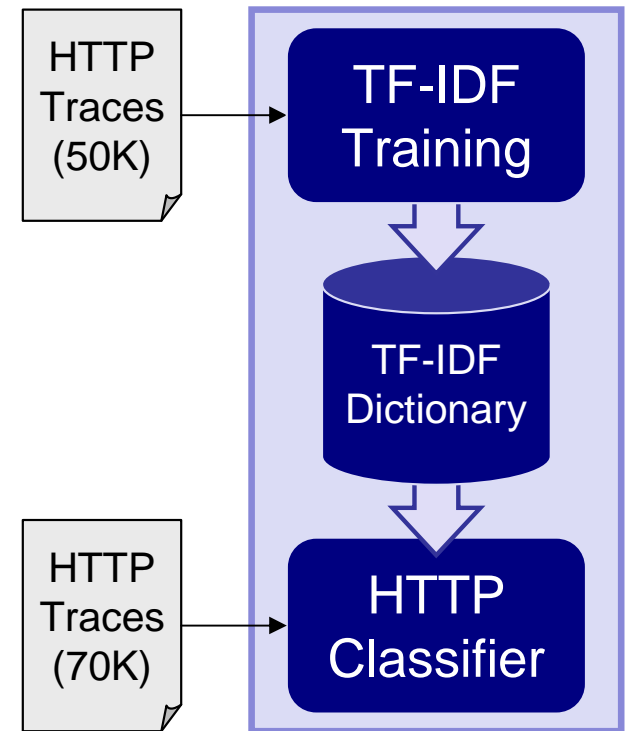
Flow Example

```
GET /eH/first_str/2hFnull6/oixsotcwrseamgit2/38PrR_Lkmmzo.htm
Host: www.a215Een.st:15
Connection: close
Accept: */*
Accept-Charset: */q=0.4
Accept-Encoding: *
Accept-Language: boHEor-sen0, gte-ltmse4 oS, 3TeoUsHn-asrao;q=0.2, paly-wreih, 78iiqths-ar;q=0.3
Cache-Control: no-store
Client-ip: 200.91.18.159
Cookie: uciy2kleicl=%3C%21---%23odbc+++++++connect%3D%226at8h%2CHcteil%2CeHnNa%22+++++statement%3D%22drop+table+elkbO...
```

The diagram illustrates the classification of the HTTP flow. Four labels are shown in rounded rectangles: 'odbc', 'connect', 'statement', and 'drop table'. Arrows point from 'odbc' and 'connect' to the 'Cookie' line. Arrows point from 'statement' and 'drop table' to the 'Cookie' line. The 'drop table' label is highlighted in red, indicating a malicious activity.

LLNL Work Achieved 99% Accuracy

- Brian Gallagher and Tina Eliassi-Rad
- Vector approach
 - Tokenize input
 - Assign weights to tokens via TF-IDF
 - Cosine similarity for vector comparison
- Relies on a data dictionary
 - Generate term statistics during training
 - Reference statistics at runtime



Top 3 SSI Classifier Terms

Term	IDF	Weight
odbc	2.079	0.0134
statement	2.079	0.0134
--	0.988	0.0126

Top 3 OS Commanding Classifier Terms

Term	IDF	Weight
..	1.386	0.0057
dir	2.079	0.0053
/c	2.079	0.0051



Equations

- Term-Frequency, Inverse Document Frequency
 - TF: How often does each term appear in an attack?
 - IDF: How specific is the term to an attack?

$$tfidf(t, d) = \underbrace{\frac{\text{count}(t, d)}{\sum_{v \in d} \text{count}(v, d)}}_{\text{Term Frequency}} \cdot \underbrace{\log \frac{|D|}{|\{d_j : t \in d_j\}|}}_{\text{Inverse Document Frequency}}$$

- Cosine Similarity
 - Vector dot product to estimate angle between input and attack

$$\text{sim}_{\cos}(a, R) = \frac{\vec{a} \cdot \vec{R}}{\|\vec{a}\| \cdot \|\vec{R}\|} = \frac{\sum_{t \in a \cap R} tfidf(t, a) \cdot tfidf(t, R)}{\sqrt{\sum_{t \in a} tfidf(t, a)^2} \cdot \sqrt{\sum_{t \in R} tfidf(t, R)^2}}$$



Equations for Programmers

$$\text{score}[\text{classifier}] = \frac{\sum_{t \in a \cap R} \frac{\text{count}[t]}{\# \text{input terms}} \cdot \text{idf}[t] \cdot \text{ClassLabelTfidf}[\text{classifier}][t]}{\sqrt{\sum_{t \in a} \left(\frac{\text{count}[t]}{\# \text{input terms}} \cdot \text{idf}[t] \right)^2} \cdot \text{DocMagnitude}[\text{classifier}]}$$

Equations for Programmers

The diagram illustrates the equation for calculating the score of a classifier based on TF-IDF values. The equation is presented as follows:

$$\text{score}[\text{classifier}] = \sum_{t \in a \cap R} \left(\frac{\text{count}[t]}{\# \text{input terms}} \cdot \text{idf}[t] \right) \cdot \text{ClassLabelTfidf}[\text{classifier}][t]$$

The equation is annotated with several components:

- Count each term in Input:** Points to the $\frac{\text{count}[t]}{\# \text{input terms}}$ term in the summation.
- Lookup IDF for term in dictionary:** Points to the $\text{idf}[t]$ term in the summation.
- Lookup weight for term in dictionary:** Points to the $\text{ClassLabelTfidf}[\text{classifier}][t]$ term.
- Scale based on TF-IDFs found by ALL classifiers:** Points to the square root term $\sqrt{\sum_{t \in a} \left(\frac{\text{count}[t]}{\# \text{input terms}} \cdot \text{idf}[t] \right)^2}$.
- Adjust based on weight of classifier:** Points to the $\text{DocMagnitude}[\text{classifier}]$ term.

Equations for Programmers

Lookup IDF for term in dictionary

Count each term in Input

Lookup weight for term in dictionary

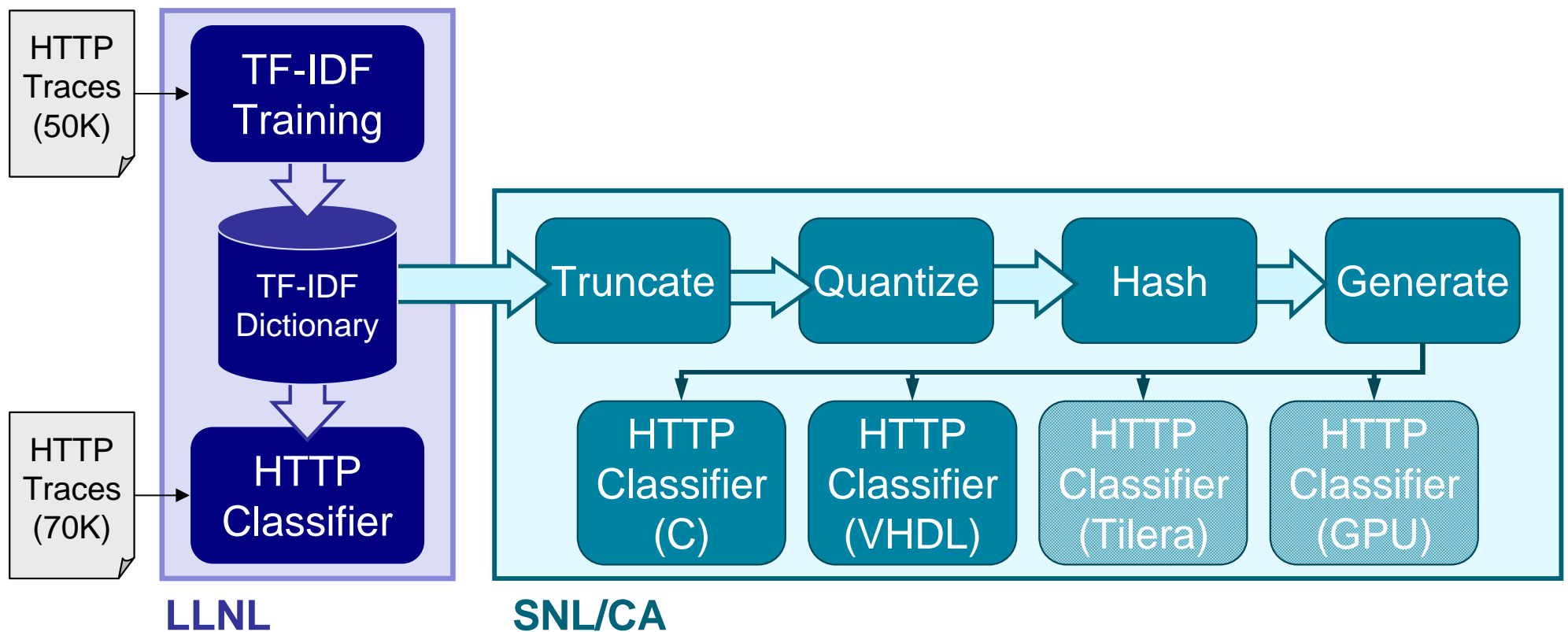
Adjust based on weight of classifier

Scale based on TF-IDFs found by ALL classifiers

$$\text{score}[\text{classifier}] = \sum_{t \in \text{input terms}} \frac{\text{count}[t]}{\# \text{input terms}} \cdot \text{idf}[t] \cdot \text{ClassLabelTfidf}[\text{classifier}][t]$$
$$\sqrt{\sum_{t \in \text{input terms}} \left(\frac{\text{count}[t]}{\# \text{input terms}} \cdot \text{idf}[t] \right)^2} \cdot \text{DocMagnitude}[\text{classifier}]$$

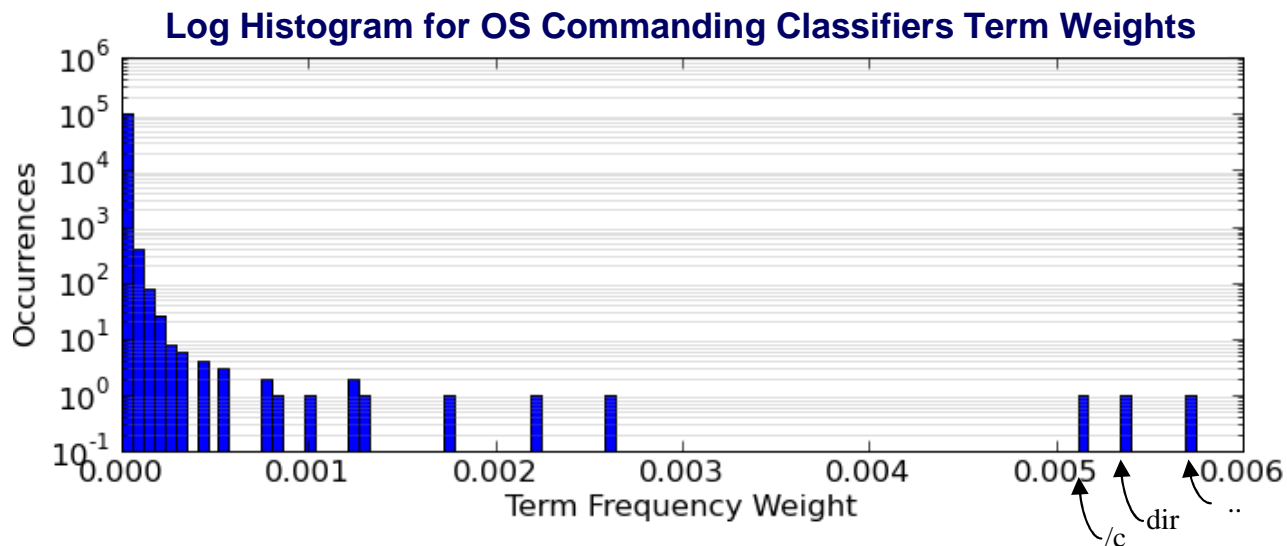
The diagram illustrates the TF-IDF equation for calculating a classifier score. It shows the standard equation with a normalization term crossed out. Annotations explain each part: 'Lookup IDF for term in dictionary' points to the $\text{idf}[t]$ term; 'Count each term in Input' points to the $\frac{\text{count}[t]}{\# \text{input terms}}$ term; 'Lookup weight for term in dictionary' points to the $\text{ClassLabelTfidf}[\text{classifier}][t]$ term; 'Adjust based on weight of classifier' points to the $\text{DocMagnitude}[\text{classifier}]$ term; and 'Scale based on TF-IDFs found by ALL classifiers' points to the normalization term $\sqrt{\sum_{t \in \text{input terms}} \left(\frac{\text{count}[t]}{\# \text{input terms}} \cdot \text{idf}[t] \right)^2}$.

The Path to Embedded Hardware

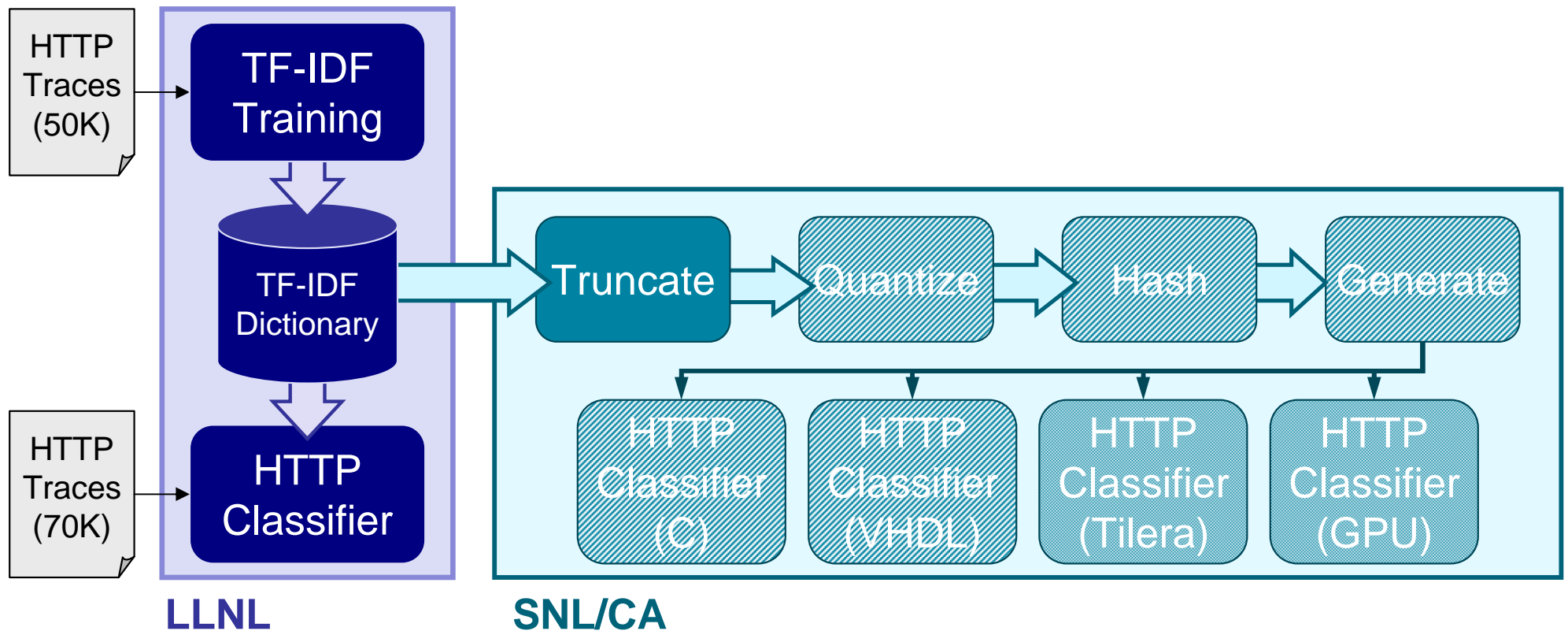


Dictionary Observations

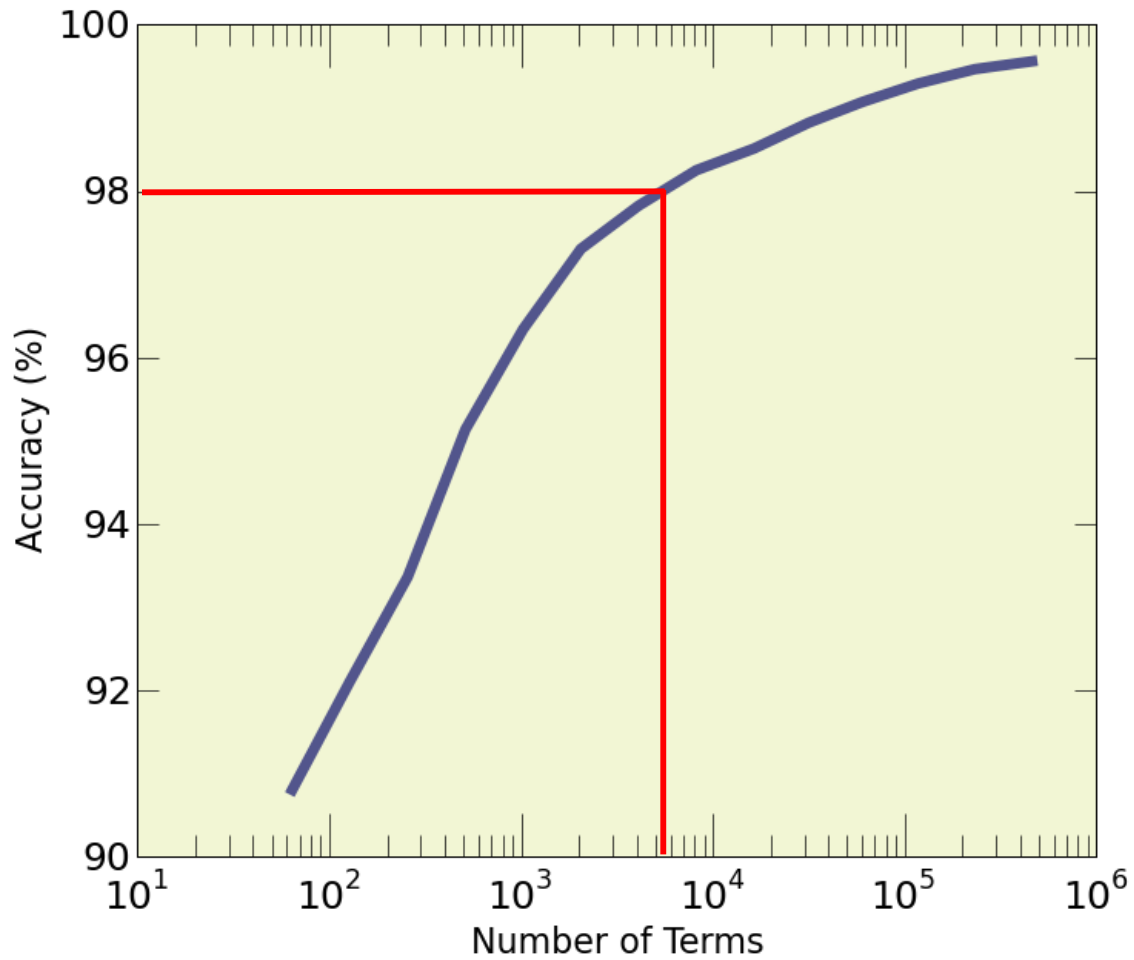
- Many terms in the dictionary
 - 1.8M terms (46MB text, 128MB data)
 - Many terms are junk (“rv:0.7.8”), but they also get very low weight
- Data values are not very diverse
 - Total unique values is < 2% of population
 - Eg: OSC Classifier has 102K terms, but only 415 unique weights



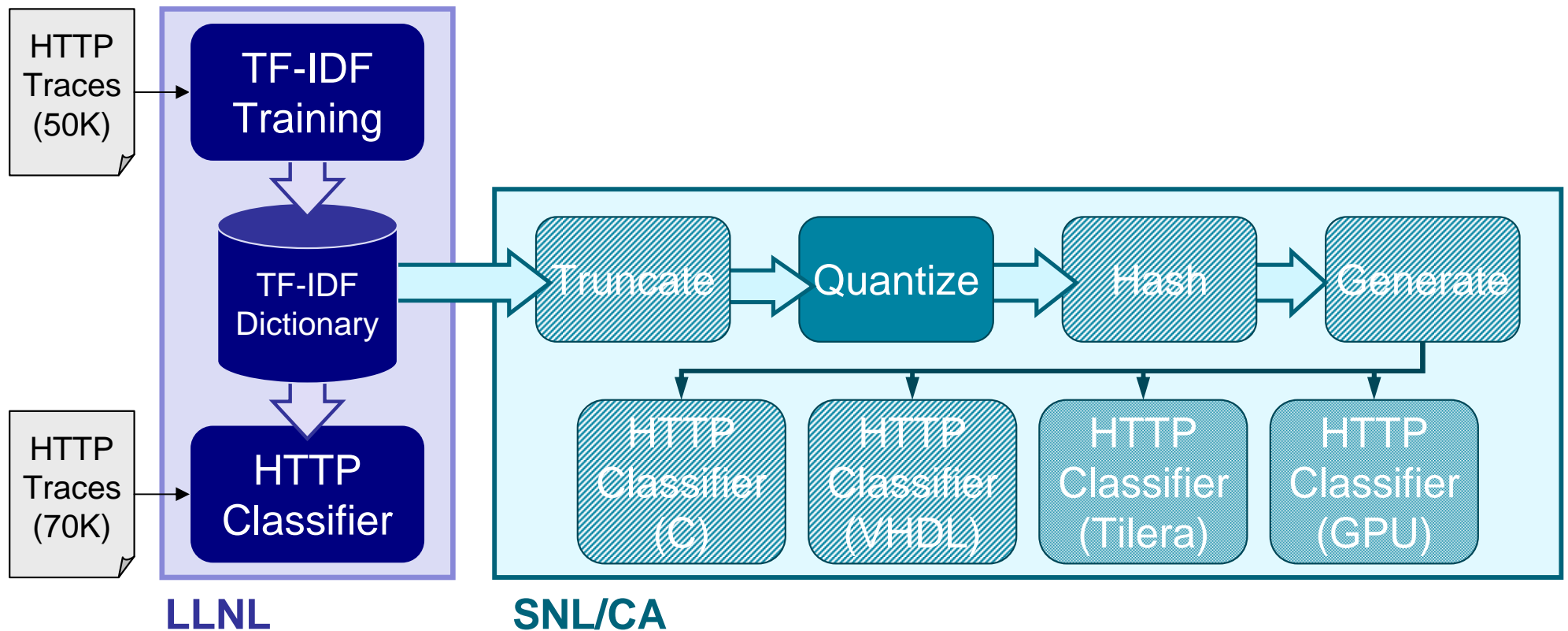
The Path to Embedded Hardware



Easy: Truncate the Dictionary

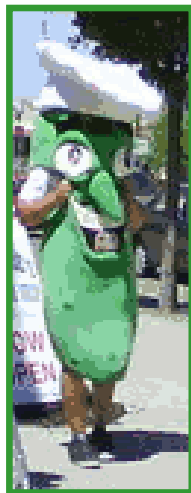


The Path to Embedded Hardware



Quantize Dictionary Data Values

- How accurate do data values in dictionary need to be?
- Does IDF("ODBC") = 0.500001 give more accurate results than..
 - 0.500002? 0.488886? 0.03?
- Experiment:
 - Reduce unique data values in dictionary, measure accuracy impact



256 Colors



64 Colors



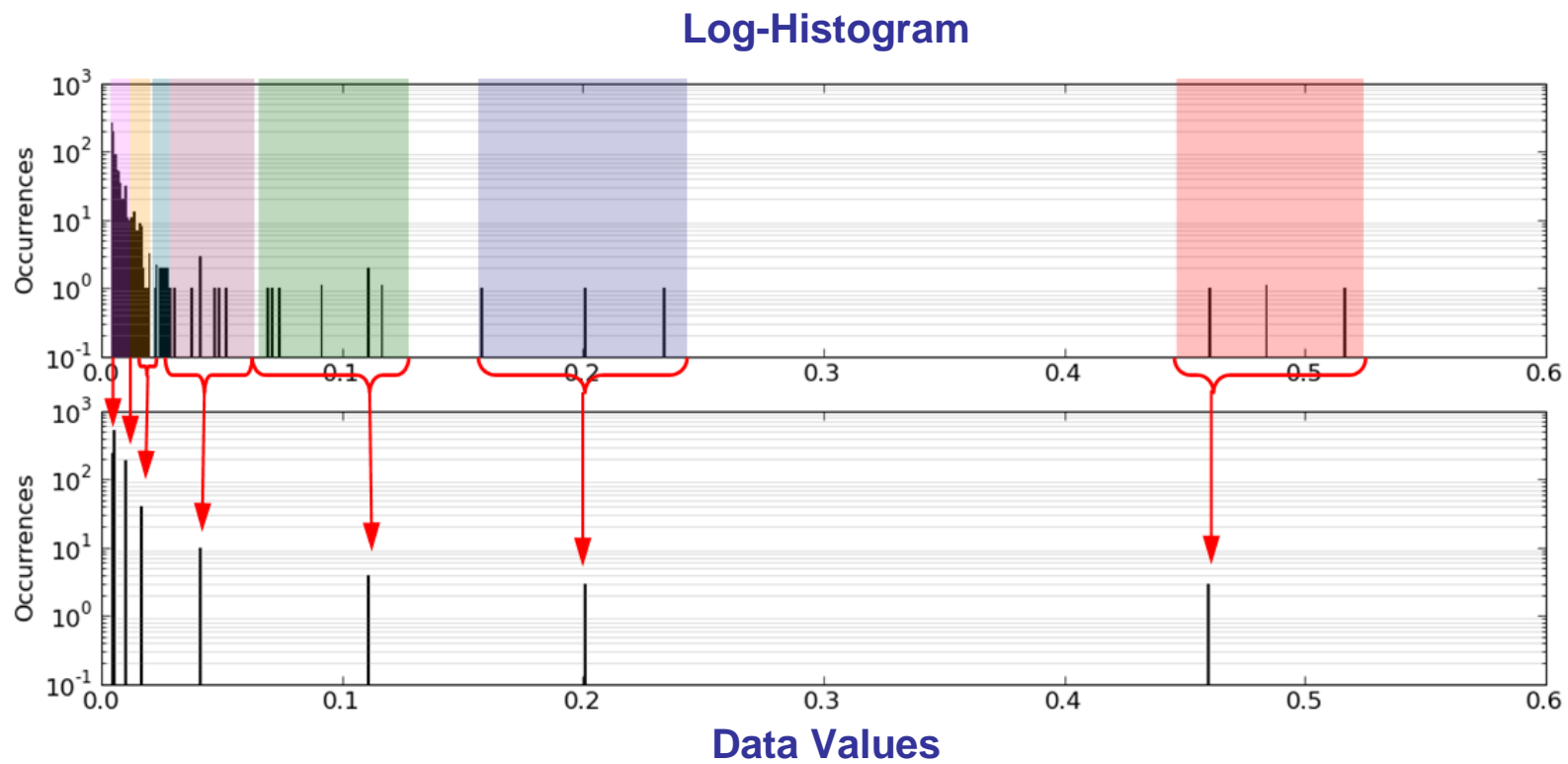
16 Colors



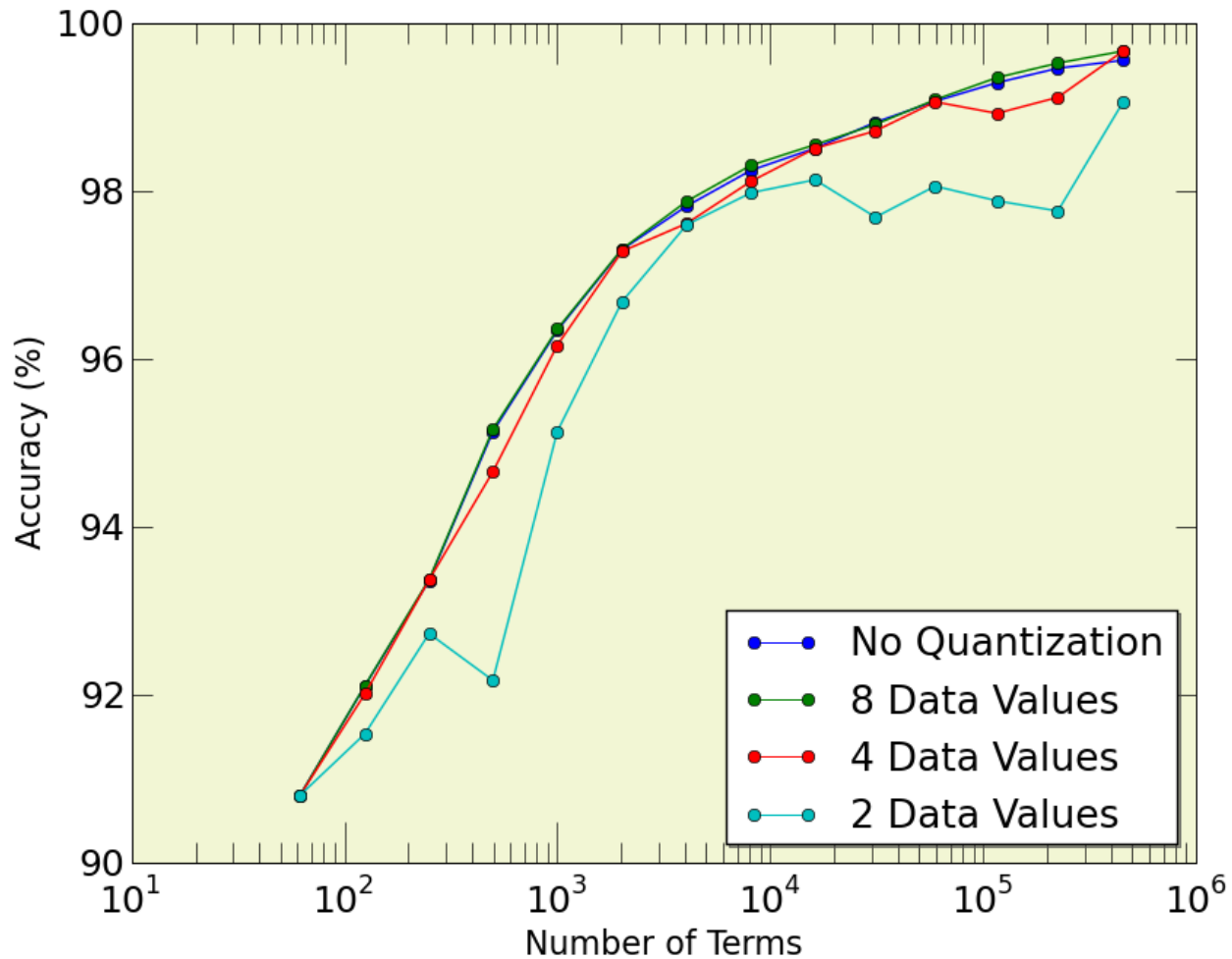
2 Colors



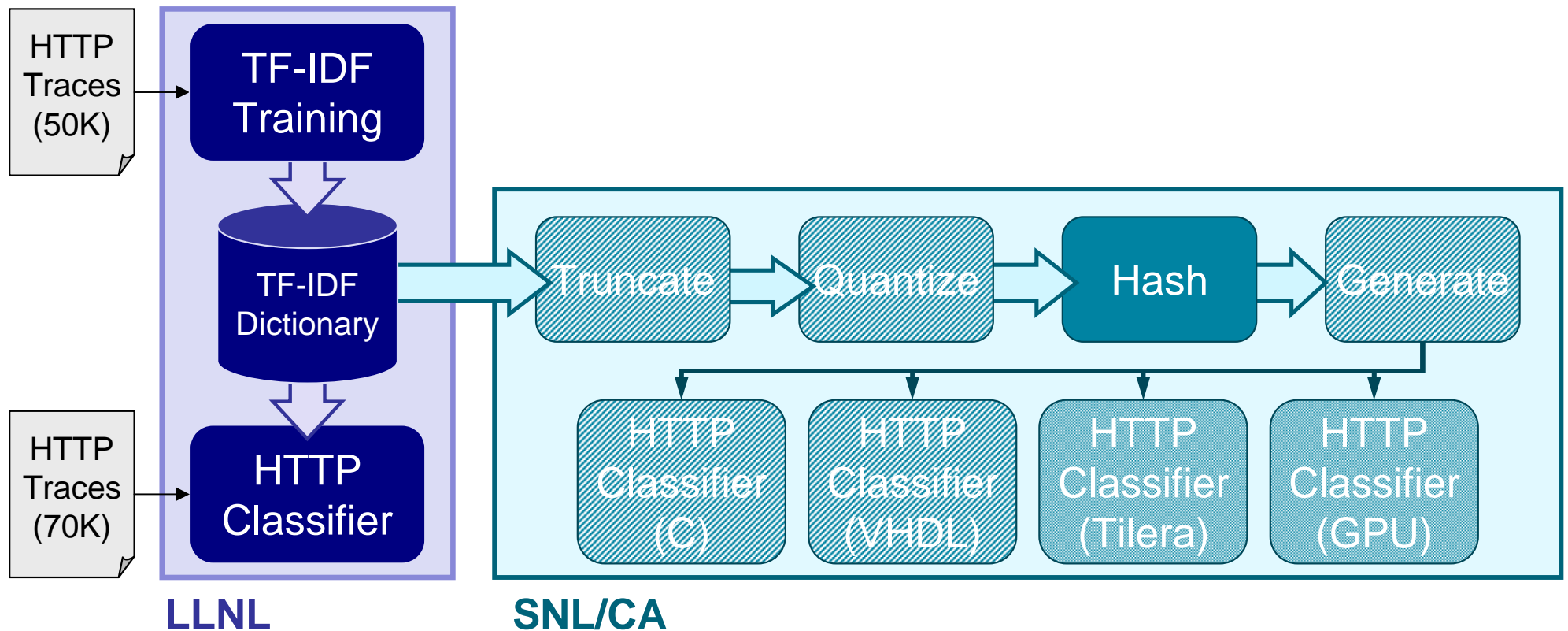
Re-Quantizing Data



Quantization Impact on End Accuracy

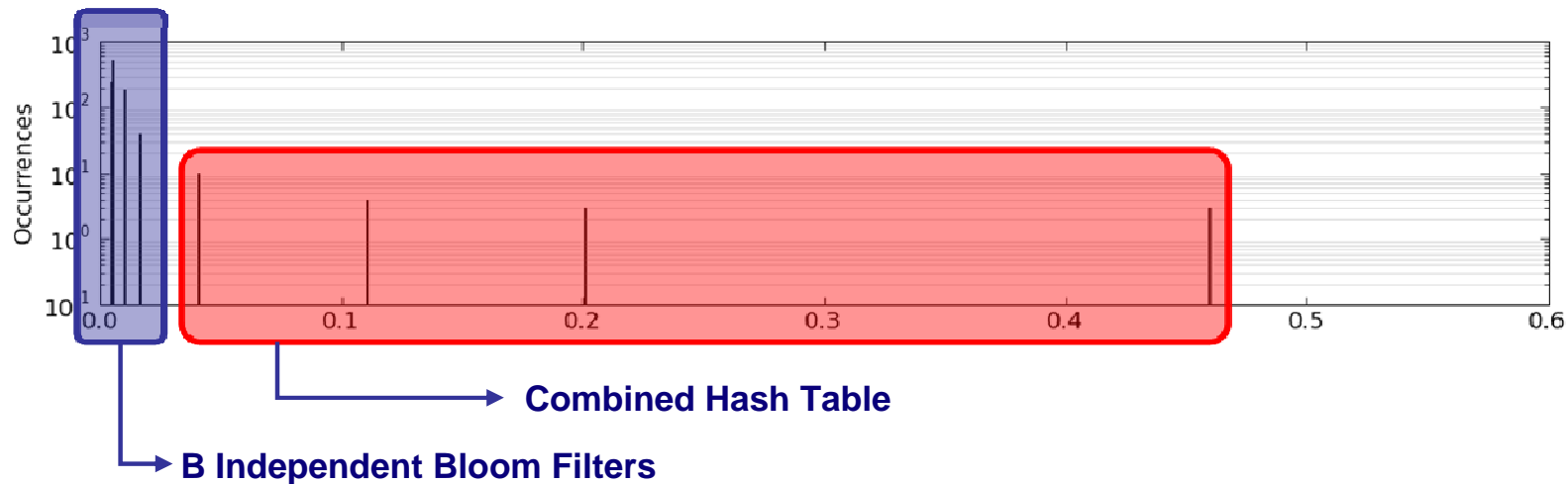


The Path to Embedded Hardware

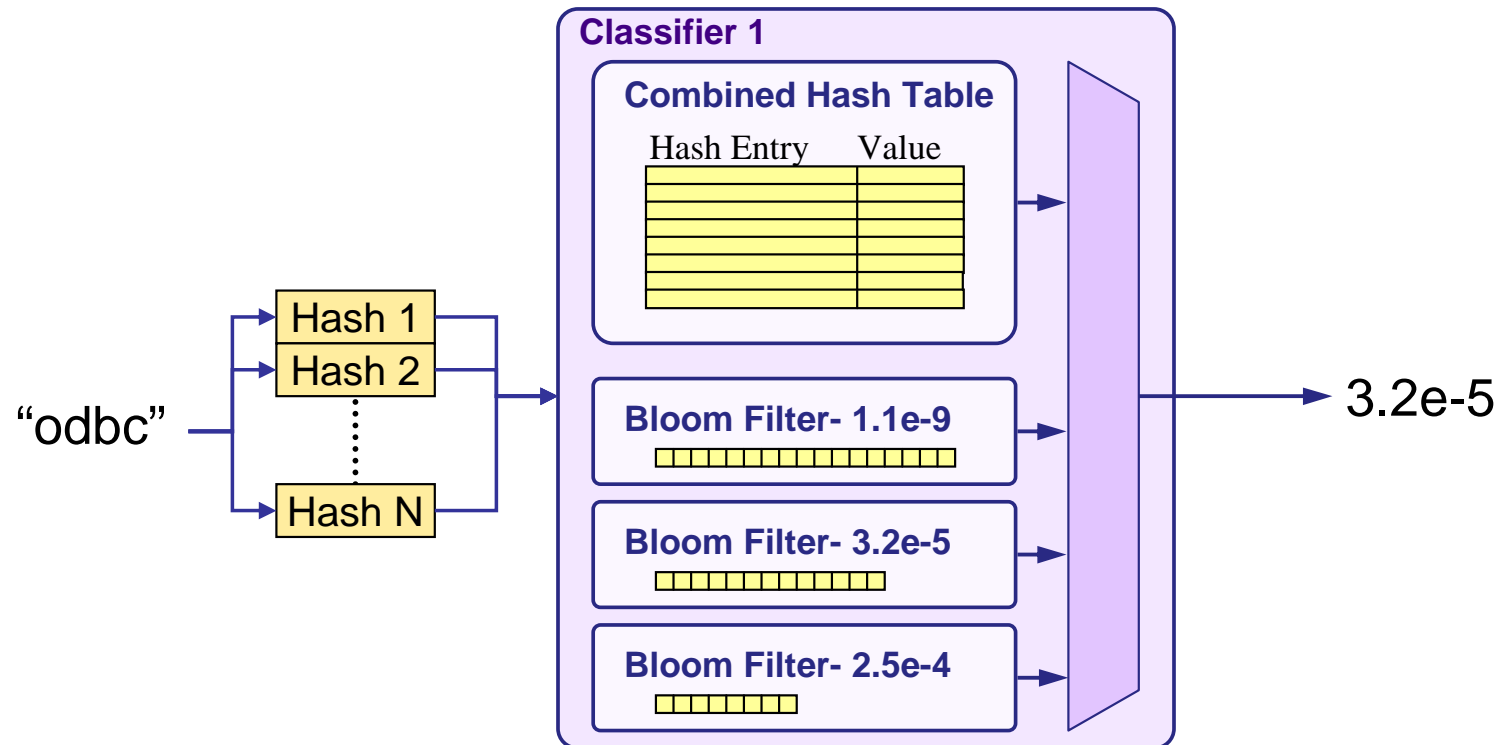


Hashing Tricks

- Small sets: combine into a single hash table
 - Brute-force packing sufficient for small tables
- Large sets: Array of Bloom filters
 - Bloom filters: space-efficient way to determine set membership
 - No false negatives, but can have false positives

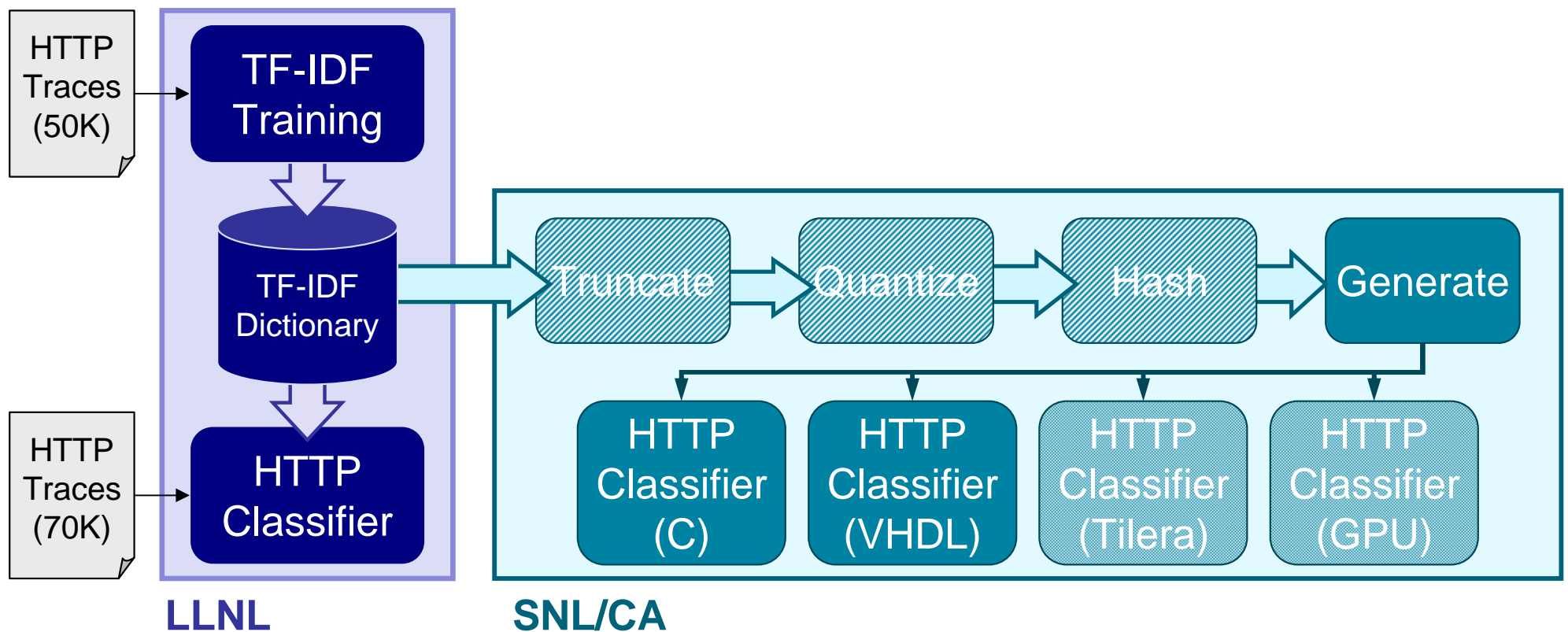


Hashing Replaces Dictionary



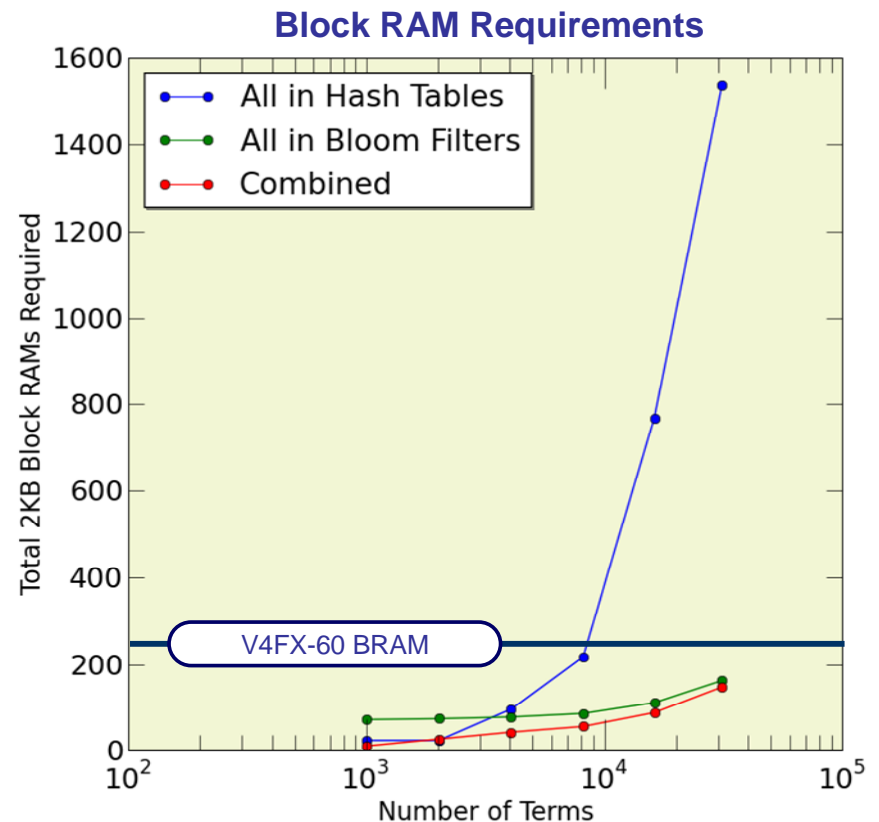
For 2KB Memory Block:
256 Hash table entries
~1K Bloom Filter members

The Path to Embedded Hardware

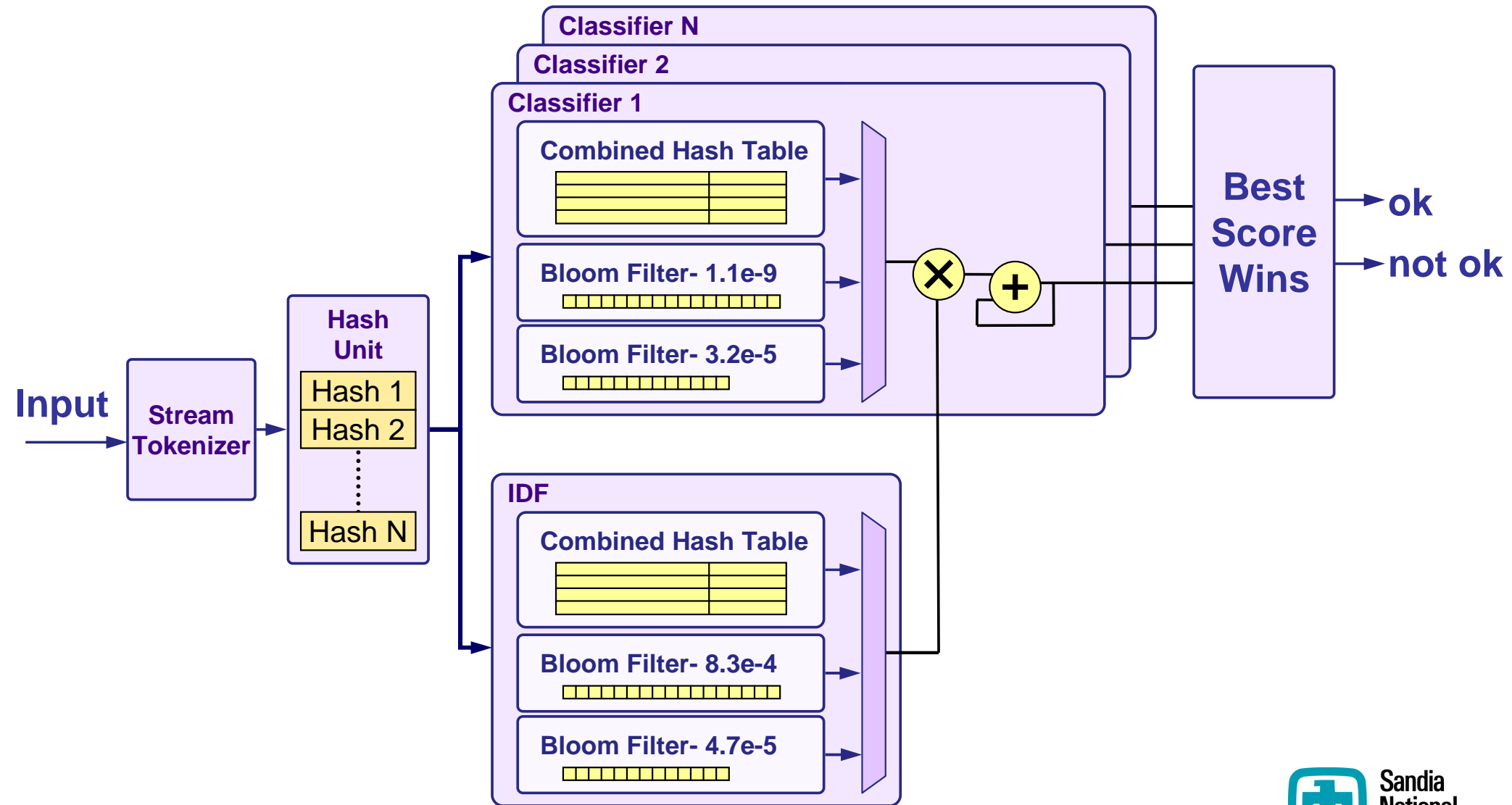


Generating Hardware

- Implemented flexible hardware design
 - Perl script converts data to parameters
- Piecewise testing
 - Full design in simulation software
 - Testing on new Virtex5 board
- Estimated speeds
 - 140MHz, >100MB/s
 - Bottleneck stream tokenizer



Hardware Data Flow





Summary

- Adapted an HTTP classifier to embedded platforms
 - Confirmed and took advantage of wiggle room in dictionary
 - Hybrid approach to hashing works well
- Relevant in other classification applications
 - TF-IDF/Cosine Similarity is a standard approach
- Ongoing/Future Work
 - Finish out a demo system by the end of FY
 - Investigate port to Tileria and GPUs