A Platform for Large-Scale Machine Learning on Web Design

Arvind Satyanarayan
SAP Stanford Graduate Fellow
Dept. of Computer Science
Stanford University
353 Serra Mall
Stanford, CA 94305 USA
arvindsatya@cs.stanford.edu

Maxine Lim
Dept. of Computer Science
Stanford University
353 Serra Mall
Stanford, CA 94305 USA
maxinel@stanford.edu

Scott R. Klemmer
Dept. of Computer Science
Stanford University
353 Serra Mall
Stanford, CA 94305 USA
srk@cs.stanford.edu

CHI'12, May 5–10, 2012, Austin, Texas, USA.
ACM 978-1-4503-1016-1/12/05.

Abstract
The Web is an enormous and diverse repository of design examples. Although people often draw from extant designs to create new ones, existing Web design tools do not facilitate example reuse in a way that captures the scale and diversity of the Web. To do so requires using machine learning techniques to train computational models which can be queried during the design process. In this work-in-progress, we present a platform necessary for doing such large-scale machine learning on Web designs, which consists of a Web crawler and proxy server to harvest and store a lossless and immutable snapshot of the Web; a page segmenter that codifies a page's visual layout; and an interface for augmenting the segmentations with crowdsourced metadata.

Author Keywords
Web design; examples; machine learning.

ACM Classification Keywords
H.5.4 [Information interfaces and presentation]: Hypertext/ Hypermedia - Theory.

Introduction
Design is difficult and designers typically spend most of their time ideating rather than executing. As a result, they often rely on examples for inspiration [4] and to

facilitate design work [8]. In the scope of Web design, the several billion pages that comprise the Web provide a large corpus of diverse design examples. While tools such as a browser's "view source" functionality [3], template galleries, and curated design repositories have begun to allow designers to harness these resources, they are limited in both usefulness and scale. For example, online repositories are manually curated by designers; therefore they usually contain a few thousand pages at most, which inadequately capture the diversity of Web designs. Furthermore, browsing designs is inefficient, typically restricted to page-by-page browsing or filtering on some basic, manually identified attributes such as color.

The unstructured, constantly changing nature of the Web, and its sheer size, pose challenges that call for an algorithmic approach to example-based Web design tools. Algorithms would not only allow these tools to pull from a larger pool of examples, but they would also enable a new class of applications. For example, consider a tool that could abstract a designer's partially-formed design idea as a search specification and extract from an extensive corpus design completions containing the input parameters. Such an application would be difficult to build with a manually curated and classified corpus.

Hard-coded, formulaic algorithms, however, would also not account for the diversity of Web design patterns, nor could they simulate non-rule-based dimensions — creativity, for example. A more suitable approach is machine learning: an algorithmic method that "learns" the characteristics of a domain, Web design in our case, from a "training set" of examples and makes predictions about these characteristics on new data.

To perform machine learning on Web designs, our training set would consist of a corpus of Web pages such that:

1. A page renders the same way every time it is requested — to generate accurate predictors, machine learning algorithms need to be trained on a constant, unchanging training set.

2. All pages within the corpus are complete copies of their online counterparts: all content they request must be stored in our corpus, and source code cannot be modified — any missing content, or modified source code, could introduce rendering or behavioral discrepancies into our corpus and corrupt classifiers trained on this data, causing them to perform poorly when run on live Web pages.

3. A representation of a page that captures the structural relationships in its visual layout is available — classifiers that rely solely on page markup would perform poorly as designers can use scripting and advanced CSS techniques to arbitrarily reposition content such that visually salient regions no longer correspond to the Document Object Model (DOM) [6].

4. The corpus is scalable and preserves the graph structure of the Web (links between pages and their resources) is key — it prevents duplicates being saved to the corpus, and the data could be useful for certain learning applications.

These requirements make it impossible to use the Web directly, as the DOM is the only representation readily available, and content changes as frequently as every page load. Thus, we need create a snapshot of the Web by

compiling a corpus that meets these requirements. There have been several projects to build such corpora [9, 7, 2] by using Web crawlers to identify and download pages and resources such as images, stylesheets and scripts.

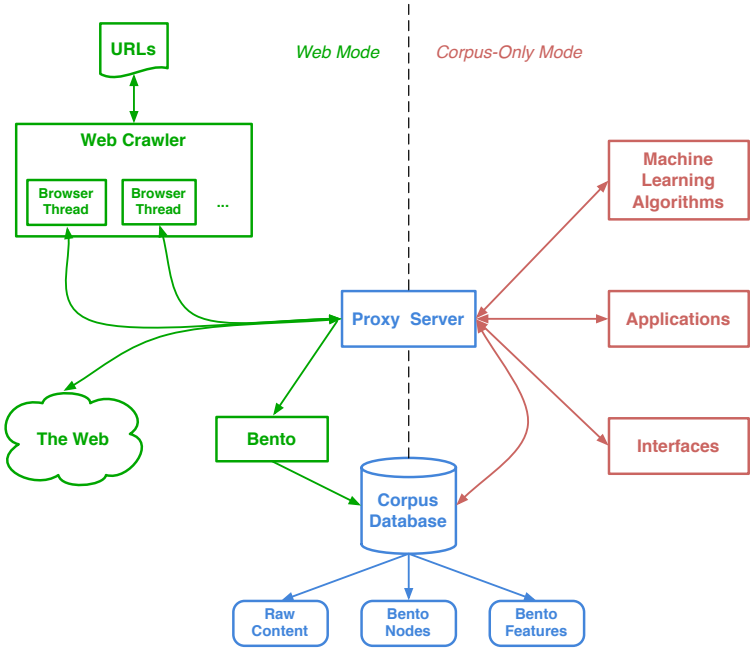

Figure 1: An overview of the system architecture.

Although Web crawlers preserve the Web's graph structure, thus scaleably building our corpus, they rely on heuristics to identify URLs to crawl [5] and may, for example, miss those embedded in CSS or requested dynamically in Javascript. This behavior would violate our second design goal. Additionally, existing crawlers treat URLs as uniquely identifying content [5], and hence only crawl them once. This does not account for cases where different content may share a common URL — for example, a Javascript file that is dynamically generated — and would again violate our second design goal. Page archivers [1] are an alternative to Web crawlers. They leverage a Web browser to download a page and package it together with an individual copy of all resources it uses. With this approach, we lose the Web's graph structure and also begin to store duplicates of resources, violating our fourth design goal. To ensure consistent offline rendering, archivers rewrite the source HTML to point to these local copies and strip any scripting, contrary to our second design goal.

To meet these design goals, we developed a platform for large-scale machine learning on Web designs. As Fig. 1 shows, we use a Web crawler and proxy server combination to build a complete corpus of raw Web page and resource content. This raw content is then processed by Bento [6], a page segmenting algorithm which produces a representation of the visual structure of a page. Finally, a crowdsourced label gathering application augments our corpus with keywords which could serve as a seed for machine learning.

The Infrastructure

Web Crawler and Proxy Server

The lightweight, multithreaded Web crawler maintains a list of URLs to crawl and then recursively loads a URL and analyzes it for additional URLs to add to the list; URLs are not treated as uniquely identifying content. Each Web crawler thread loads URLs in an embedded Webkit[1] browser, and the task of downloading content is delegated by configuring the browser to use a custom proxy server. Every request made by the browser, and every response it receives from the Internet, goes through the proxy server, which writes this information to our corpus. As URLs are no longer considered unique, to prevent the corpus from burgeoning, we implement a "content-seen" test [5] to check if it already exists in our corpus, and only saved if it does not. With this pipeline, we are guaranteed that the proxy server automatically sees every request a page makes for a resource, without any custom heuristics, and yet scalably builds our corpus.

[1] http://www.webkit.org/

The proxy server, however, simply sees a series of requests and their corresponding responses — as HTTP is a stateless protocol, it is not able to identify which page requested what resource, a problem exacerbated by multiple Web crawler threads making concurrent requests through the proxy. This relationship is crucial to preserving the Web's graph within our corpus.

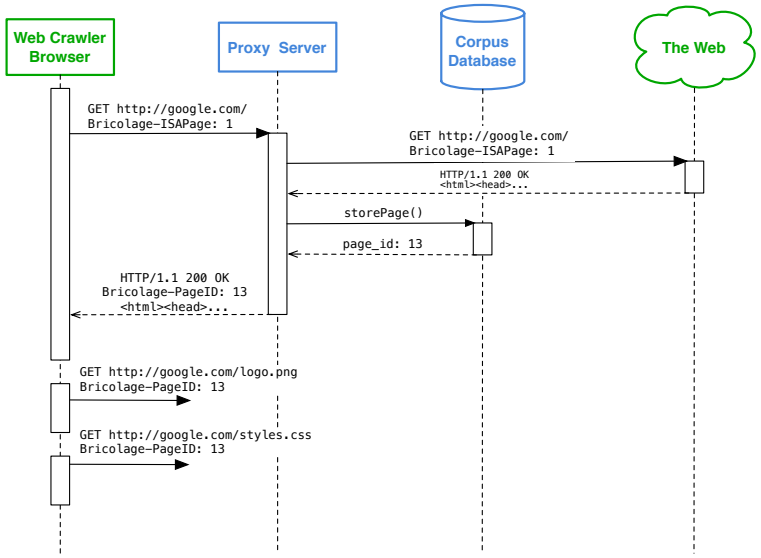

Figure 2: Custom headers are used to overcome HTTP's statelessness by identifying which page made what request. This allows us to preserve the Web's graph structure in our corpus.

To easily identify these relationships, we introduced a set of custom HTTP headers into the pipeline as shown in Fig. 2. The Web crawler requests a page and supplies the `Bricolage-ISAPage` header to identify the URL as a "page" as opposed to a "resource". When the proxy server receives the page's content from the Internet, and saves it to our corpus, a unique identifier is generated, which is transmitted back to the Web crawler's browser through the `Bricolage-PageID` header. This header is then attached to subsequent resource requests, which allows the proxy server to recreate the graph within our corpus.

As Fig. 1 shows, the system also contains a "corpus-only mode." In this mode, the proxy server does not pass requests to the Web, but rather serves content directly from our corpus; if it is not able to find the URL in the corpus, it responds with a 404 error. This mode, facilitated by adding a custom `Bricolage-FromCache` header to requests, allows consistent offline rendering without modifying any source code.

Bento: A Page Segmenting Algorithm

Each page is passed through the Bento page segmenter [6] which wraps all inline content with `<span>` tags, rearranges the DOM tree relationships to correspond to visual containment and finally augments the tree to remove nodes that do not contribute to the visual layout, and add those that do. Fig. 3 shows an example Bento tree, and highlights that all content is a leaf node in the tree, and non-leaf nodes correspond to screen space. For maximum granularity, each Bento node, and its associated style features, are stored as individual records within our corpus database.

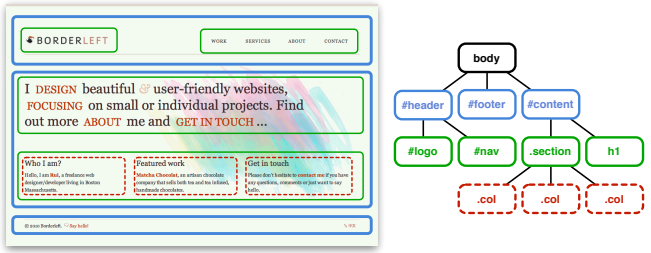

Figure 3: Bento nodes better correspond to visually salient regions of a page.

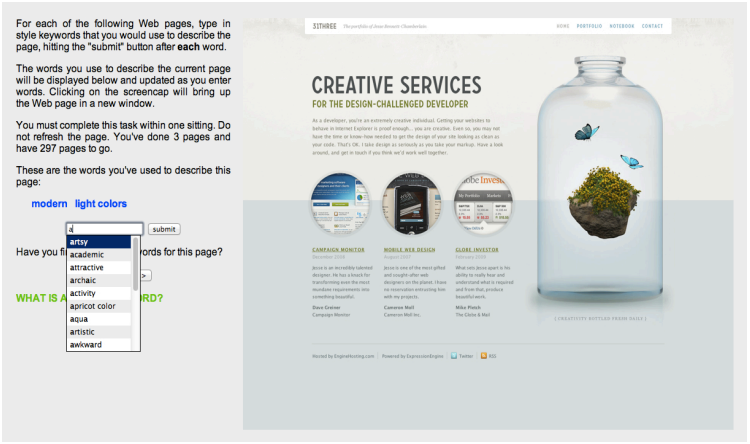

Figure 4: Label gathering interface.

Correctly and effectively collecting Web pages into a corpus provides the constant training set required for machine learning, but supplying visual, design-oriented features associated with the collected pages for the learning algorithms to use is also critical for usefulness. To allow for this functionality, labels corresponding to the features in question must be stored with the pages with which they are associated. These labels may express the stylistic features of a Web page, its content and purpose, or any other characteristics that may be of interest to designers. While labels that can be drawn from the lower-level representation of a saved Web page can be applied to the page automatically, less concrete characteristics, which may be based on a holistic visual assessment of a Web design, do not lend themselves well to automated classification. For example, automatically and accurately judging whether a design is minimalist or elegant is challenging because there does not exist a set of criteria that can be easily and methodically checked and that will also consistently result in an accurate classification. Therefore a viable solution must make consistent and accurate classifications as well as maintain the scalability that comes with automated labeling. We propose a label-collecting interface, shown in Fig. 4, that allows us to take advantage of crowdsourcing. If label application is delegated to Web designers themselves, we can attain accurate style labels; if a large enough audience is targeted, scalability can be achieved as well.

To allow designers to apply labels to designs, we built a label-collecting interface. Designers were sent to this interface and asked to submit stylistic keywords describing a number of Web pages in our corpus. The interface includes a set of directions for entering labels into the database, a screenshot of the current page, a list of keywords the designer has already applied to the page, and appropriate fields for label submission and page navigation. Upon access to the interface, a unique ID is randomly generated for the user, and this ID is maintained throughout the task. The corpus of Web pages to be labeled loads, and pages are presented to the user in random order. For each page, the user enters and submits a number of keywords to describe it. The keyword field is enhanced with a jQuery-backed autocomplete feature drawing from a dictionary of valid English keywords that have been applied to pages in the database. The dictionary is refreshed after every page entry. Upon submission each new keyword, along with the associated user ID and Web page, is entered into a SQLite database and appears on the screen so that users can track which keywords they have already entered. Keyword entries and page navigation are implemented using AJAX calls to avoid refreshing the interface page.

Discussion and Conclusion

To assess the feasibility of the label-collecting interface, we ran a preliminary trial to gather style labels for a 300-page corpus by hiring crowdsourced designers to apply labels to the pages. The interface allowed us to collect over 3,000 style labels, shown as a tag cloud in Fig. 5, that appropriately described their corresponding pages.

Figure 5: A tag cloud of the 3,000 labels collected in our preliminary trial. Notice that some style words are used more commonly than others.

However, though we were able to pay designers to perform this task for the purposes of this trial, providing such compensation does not allow for scalability. If designers had incentive to apply labels themselves, then payment would not be necessary. By building a searchable interface for our corpus based on the collected labels, we would be able to offer designers a powerful exploratory tool for inspiration unlike any design repository currently available. The usefulness of the tool would allow for scalability, since designers would have an incentive to supply labels for the underlying corpus in order to improve their experience with the interface.

The interface described is only one of the many applications that could stem from our corpus. Trained programs may eventually be able to automatically and accurately classify designs based on stylistic features. Design tools in the same vein as the searchable interface described may provide designers with relevant inspirational examples when given a partially complete design or an existing design example as input. They might even generate original designs based on a set of requirements specified by the designer. Additionally, browser extensions to crowdsource corpus-building can be developed to facilitate augmenting the corpus itself and increase the range of features it can support. To more thoroughly grasp the power of machine learning in the context of Web design, we look forward to building these applications and investigating their implications for designers.

References

[1] Mozilla archive file format. `http://maf.mozdev.org/`.

[2] Cathro, W., Webb, C., and Whiting, J. Archiving the web: The pandora archive at the national library australia. *National Library of Australia Staff Papers*, 0 (2009).

[3] Gibson, D., Punera, K., and Tomkins, A. The volume and evolution of web page templates. In *Special interest tracks and posters of the 14th international conference on World Wide Web*, ACM (2005), 830–839.

[4] Herring, S., Chang, C., Krantzler, J., and Bailey, B. Getting inspired!: understanding how and why examples are used in creative design practice. In *Proceedings of the 27th international conference on Human factors in computing systems*, ACM (2009), 87–96.

[5] Heydon, A., and Najork, M. Mercator: A scalable, extensible web crawler. *World Wide Web 2*, 4 (1999), 219–229.

[6] Kumar, R., Talton, J., Ahmad, S., and Klemmer, S. Bricolage: Example-based retargeting for web design. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, ACM (2011), 2197–2206.

[7] Lampos, C., Eirinaki, M., Jevtuchova, D., and Vazirgiannis, M. Archiving the greek web. In *4th International Web Archiving Workshop (IWAW04)* (2004).

[8] Lee, B., Srivastava, S., Kumar, R., Brafman, R., and Klemmer, S. Designing with interactive example galleries. In *Proceedings of the 28th international conference on Human factors in computing systems*, ACM (2010), 2257–2266.

[9] Mohr, G., Stack, M., Rnitovic, I., Avery, D., and Kimpton, M. Introduction to heritrix. In *4th International Web Archiving Workshop* (2004).