UNIVERSITY OF CAMBRIDGE

SIGCOMM BIG-DAMA 2018

Data Analytics Service Composition and Deployment on Edge Devices

Jianxin Zhao, Tudor Tiplea, Richard Mortier, Jon Crowcroft, Liang Wang

University of Cambridge

August 2018

Motivation

- With social awareness of privacy and personal data rapidly rising, it becomes a pressing and challenging societal issue to both keep personal data private and benefit from the data analytics power of machine learning (ML) techniques.
- The currently popular cloud-based ML services are known to associate with issues such as communication cost, latency, and personal data privacy.
- To avoid those costs, reduce latency in data processing, and minimise the raw data revealed to service providers, many AI and ML services could be partly deployed on users’ devices at the Internet edge rather than putting everything on the cloud.
- Use cases: automatic cars, personal data analytics in home, DNN on stick, caching, etc.

Motivation

Cloud-based Deployment Systems:

- Clipper, Tensorflow Serving
- “Function-as-a-Service”: AWS Lambda, OpenWhisk, ...
- Application-specific: LASER, NoScope, etc.

Edge Deployment:

- EdgeML, Azure IoT edge, AWS Greengrass, etc.

Motivation

Cloud-based Deployment Systems:

- Clipper, Tensorflow Serving
- “Function-as-a-Service”: AWS Lambda, OpenWhisk, ...
- Application-specific: LASER, NoScope, etc.

Edge Deployment:

- EdgeML, Azure IoT edge, AWS Greengrass, etc.

Two Challenges:

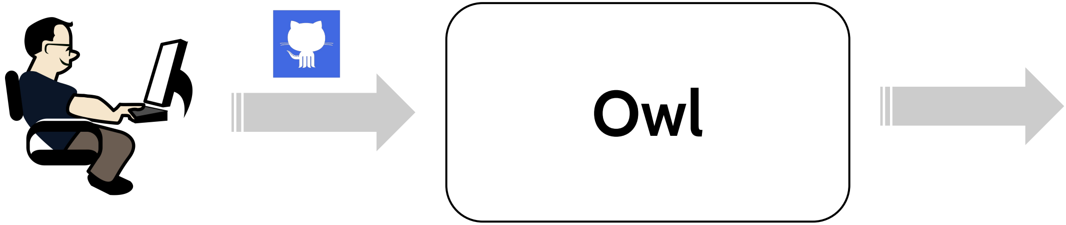
- **How to re-use existing computation code?**
- **How to easily deploy same code on different devices?**

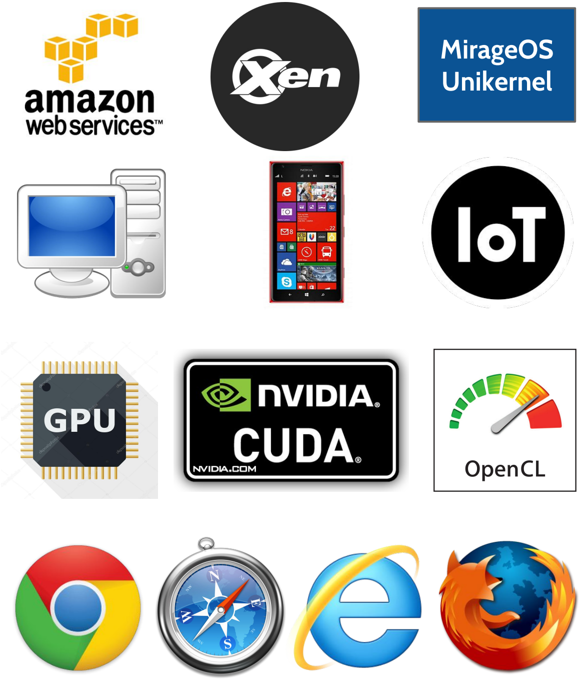
Owl Numerical Library

- An experimental and above all scientific computing system.
- Designed in functional programming paradigm.
- Goal: as concise as Python yet as fast as C, and safe.
- A comprehensive set of classic numerical functions.
- A fundamental tooling for modern data analytics (ML & DNN).
- Native support for algorithmic differentiation, distributed & parallel computing, and GPGPU computing.

Vision Beyond Research Prototype

Write code once, then deploy it everywhere …

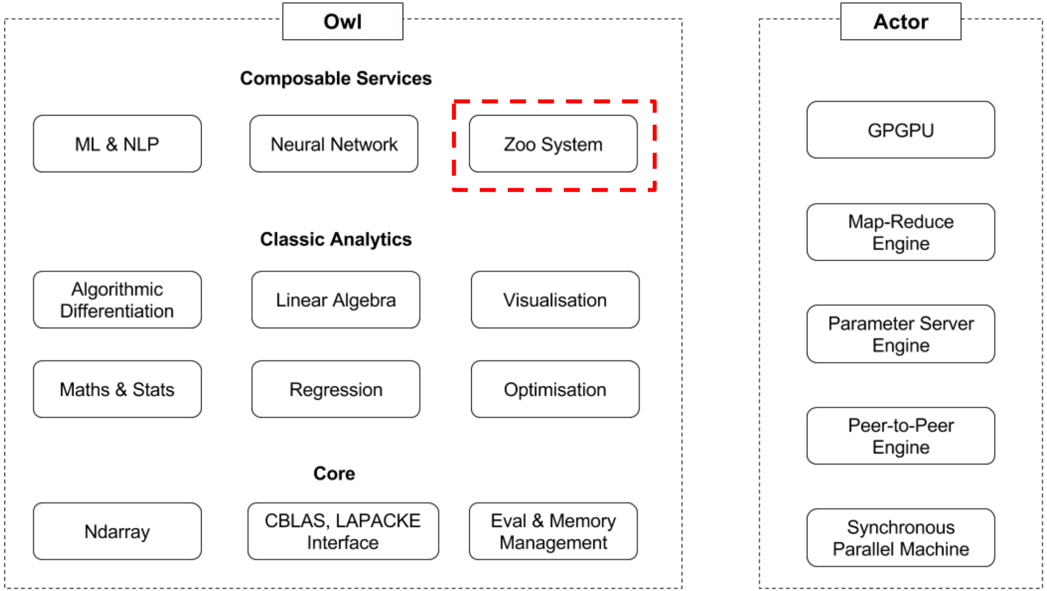




Owl system provides us a complete set of tooling from the powerful numerical supports in development to the deployment on various platforms.

Reference: Wang, Liang. "Owl: A General-Purpose Numerical Library in OCaml." arXiv preprint arXiv:1707.09616 (2017).

Owl Numerical Library



Designed and Developed by Dr. Liang Wang

Owl + Actor = Distributed & Parallel Analytics

Owl provides numerical backend; whereas Actor implements the mechanisms of distributed and parallel computing. Two parts are connected with functors.

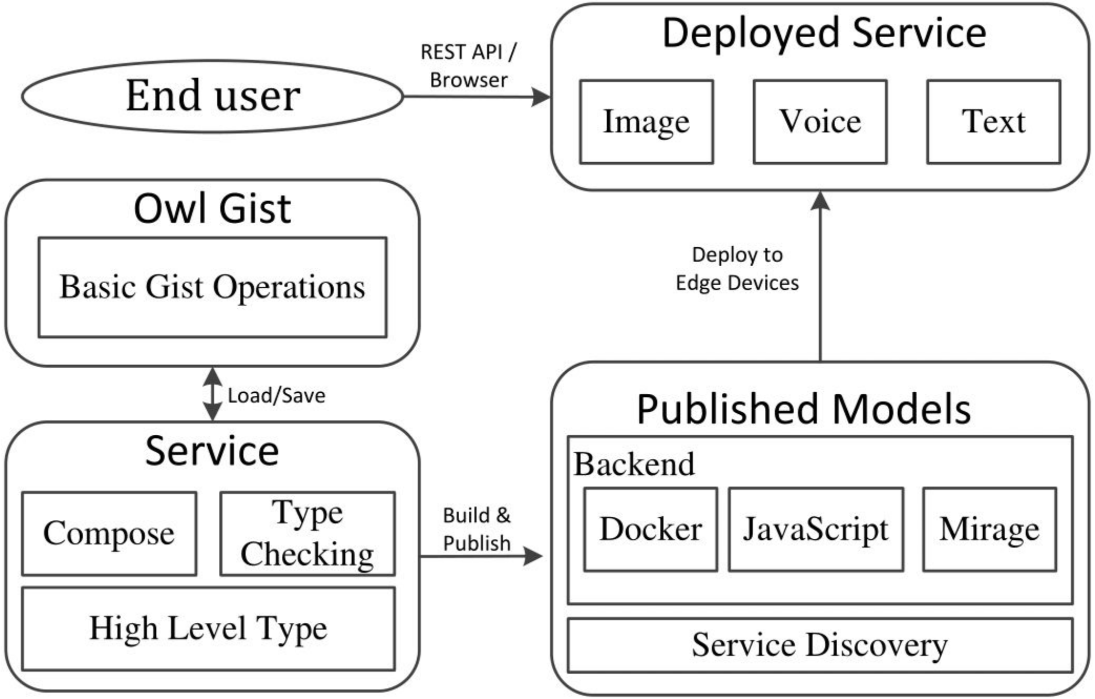
Various system backends allows us to write code once, then run it from cloud to edge devices, even in browsers.

Same code can run in both sequential and parallel mode with Actor engine.

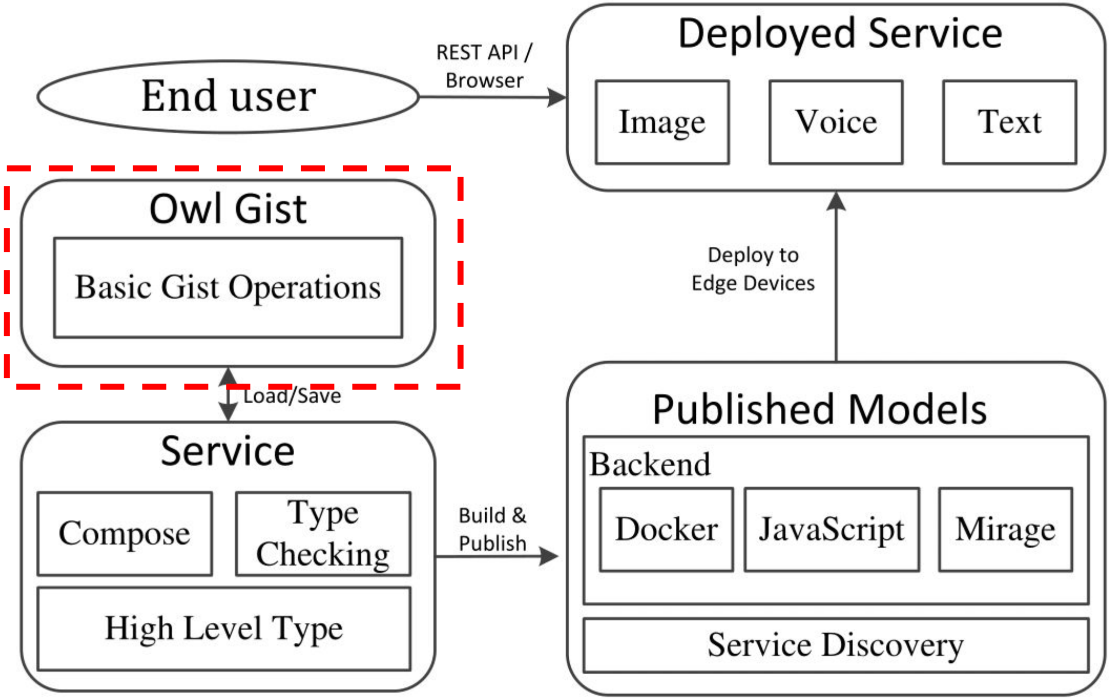
Reference: Wang, Liang. "Owl: A General-Purpose Numerical Library in OCaml." arXiv preprint arXiv:1707.09616 (2017).

UNIVERSITY OF CAMBRIDGE

Zoo System

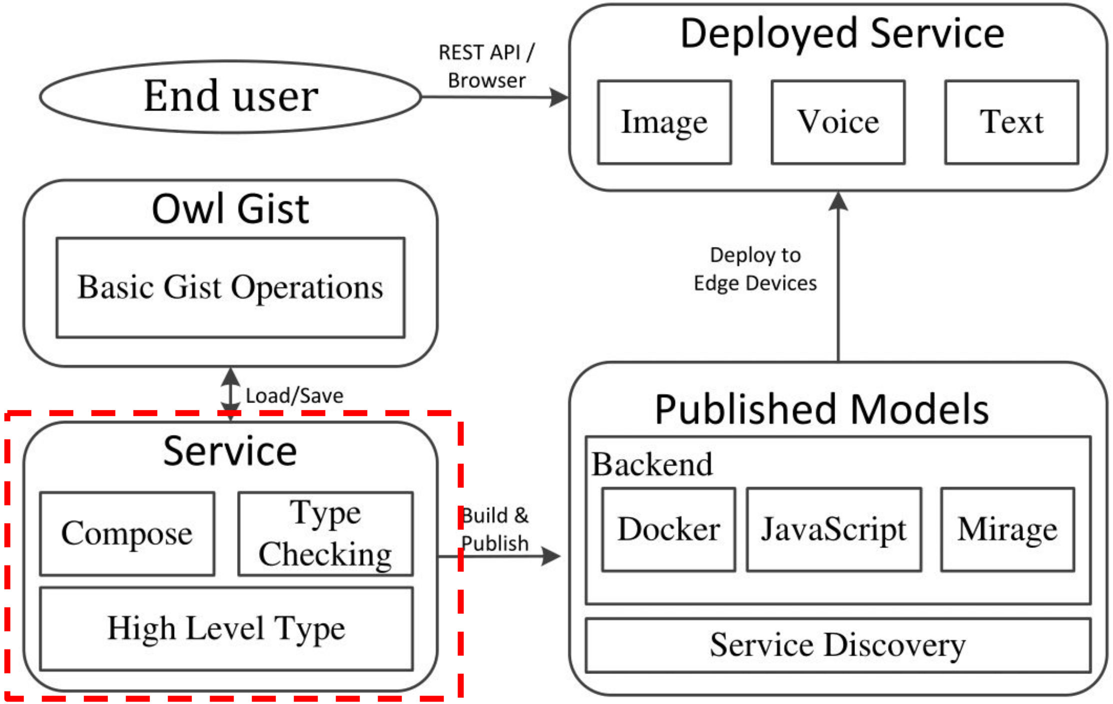

End user
REST API / Browser
Deployed Service
Image
Voice
Text
Owl Gist
Basic Gist Operations
Load/Save
Deploy to Edge Devices
Service
Compose
Type Checking
High Level Type
Build & Publish
Published Models
Backend
Docker
JavaScript
Mirage
Service Discovery

Zoo System



UNIVERSITY OF CAMBRIDGE

Zoo System



UNIVERSITY OF CAMBRIDGE

Zoo System: Service

- New abstraction: **service** (function)
  - **Gist** : list of gist ids this service requires
  - **Types**: parameter types of this service
  - **Dependency Graph**: a graph structure that contains information about how the service is composed. Each node consists of gist's name, id, and number of parameters
- Operations of service:
  - create ($): creates a dict of services from gist id
  - get ($~): get a service from a dict by name
  - compose ($>): combine multiple services into one

```
type t = {
 mutable gists : string array;
 mutable types : string array;
 mutable graph : (string * string *
int) Owl_graph.node;
}
val ( $  ) : string -> (string, t)
Hashtbl.t
val ( $~ ) : (string, t) Hashtbl.t ->
string -> t
 val ( $> ) : ?name:string -> t list ->
t -> t list
```

Zoo System: Types

```
type _ img =
 | PNG : string -> png  img
 | JPG : string -> jpeg img
 | PPM : string -> ppm  img

type _ text =
 | ENT : string -> en text
 | FRT : string -> fr text

let string_of_img (type el) (x:el img) =
 match x with
 | PNG a -> a
 | JPG a -> a
 | PPM a -> a
```

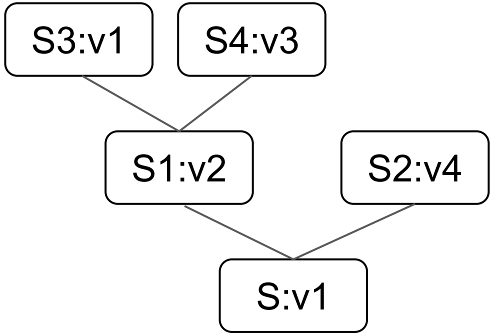
```
type z =
 | Z_string    of string
 | Z_float     of float
 | Z_int       of int
 | Z_bytes     of bytes
 | Z_bool      of bool
 | Z_ndarray_s of Owl.Dense.Ndarray.S.arr
 | Z_ndarray_d of Owl.Dense.Ndarray.D.arr
 | Z_png_img   of png img
 | Z_jpg_img   of jpeg img
 | Z_ppm_img   of ppm img
 | Z_en_text   of en text
 | Z_fr_text   of fr text
 | Z_en_voice  of en voice
 | Z_fr_voice  of fr voice
 | Z_list      of z list
 | Z_array     of z array
```

Zoo System: Compose

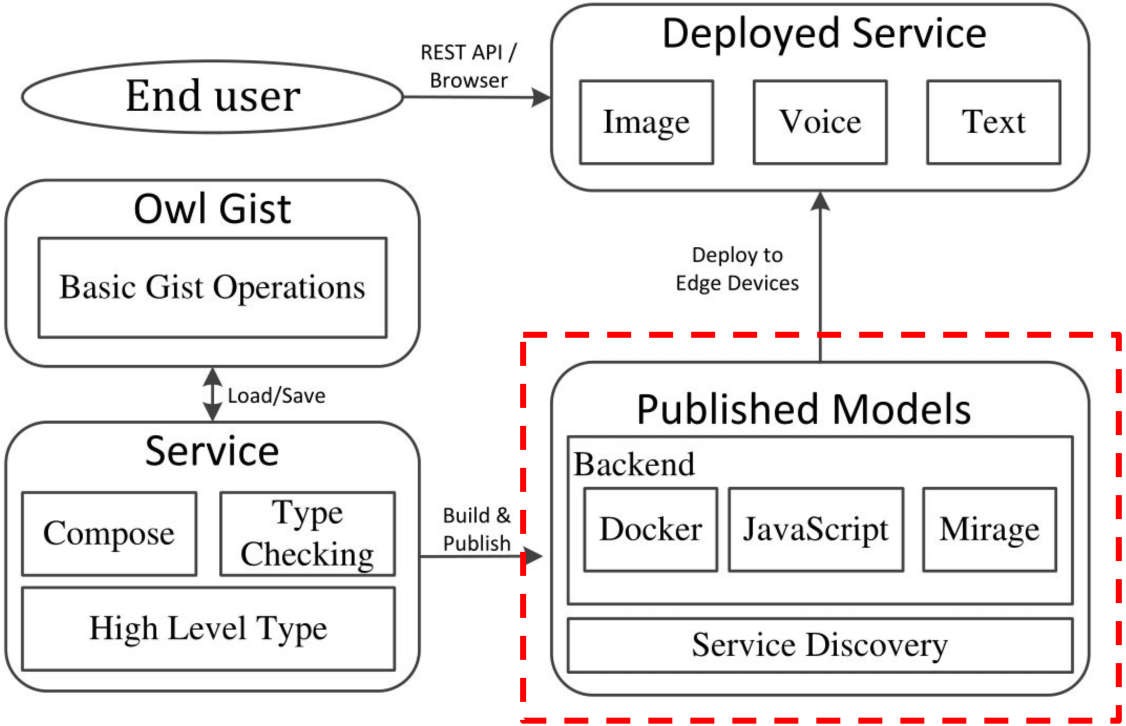
```
(* Basic Usage *)
#zoo
"e7d8b1f6fbe1d12bb4a769d8736454b9?vid=fc56e09e08978f62e4f1958272
53abbda4c2b40e" (* LoadImage *)
#zoo "41380a6baa6c5a37931cd375d494eb57?tol=0" (* SqueezeNet  *)

(* Service Compose *)
open Owl_zoo_service
open Owl_zoo_utils
let ss1 = $ "aa36ee2c93fad476f4a46dc195b6fd89";;
let s1  = ss1 $~ "Squeezenet.infer"
let s2  = ss1 $~ "Squeezenet.to_json"
let ss2 = $ "7f32af9c1691fbfcf4f4340bd3780ee8";;
let s3  = ss2 $~ "Word_count.word_count"
let new_service = [s1] $> s2 $> s3
```

- Use Gist as a source of services
- Define a service by composing existing ones
- Version and dependency control mechanisms vs. OPAM



Zoo System



UNIVERSITY OF CAMBRIDGE

Zoo System: Build

```
type backend =
  | CREST  of CREST.backend_typ
  | JS     of JS.backend_typ
  | Mirage of Mirage.backend_typ
val preprocess : backend -> string -> unit
val gen_build_files : backend -> string -> string -> unit
val build_exec : backend -> string -> unit
val postprocess : backend -> string -> unit
```

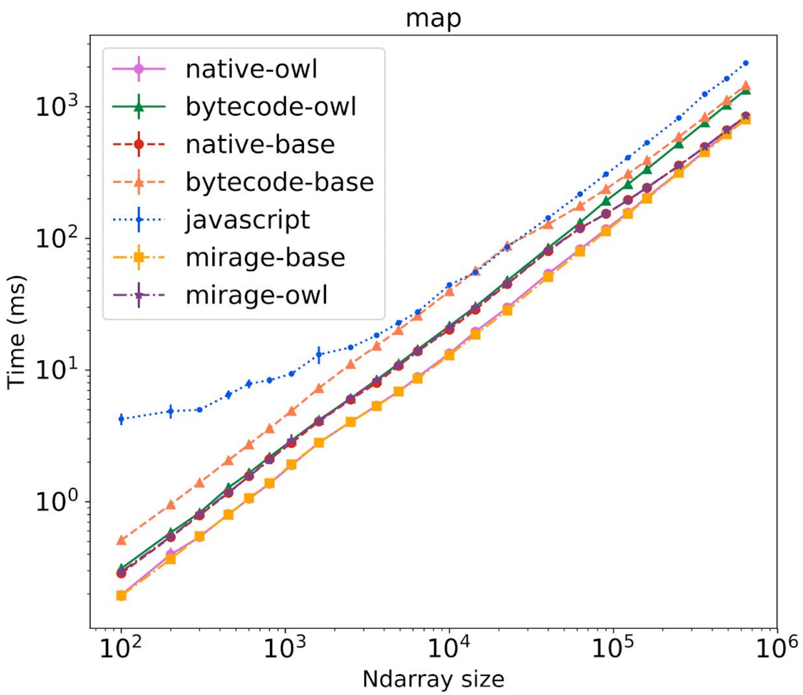
```
open Owl_zoo
open Owl_zoo_build
let gist = "aa36ee2c93fad476f4a46dc195b6fd89" in
let backend = CREST {dname = "alice/squeeznet:latest"} in
gist $@ backend
```

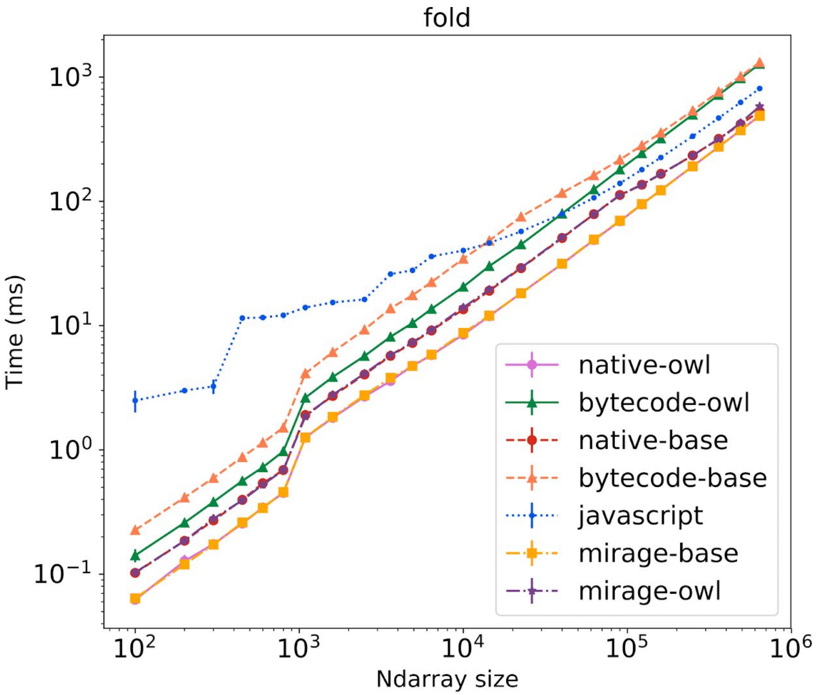
- Service development should be separated from its deployment.

```
{ "Squeezenet.infer": "png_img -> ndarray",
 "Squeezenet.to_json": "ndarray -> en_text" }
```

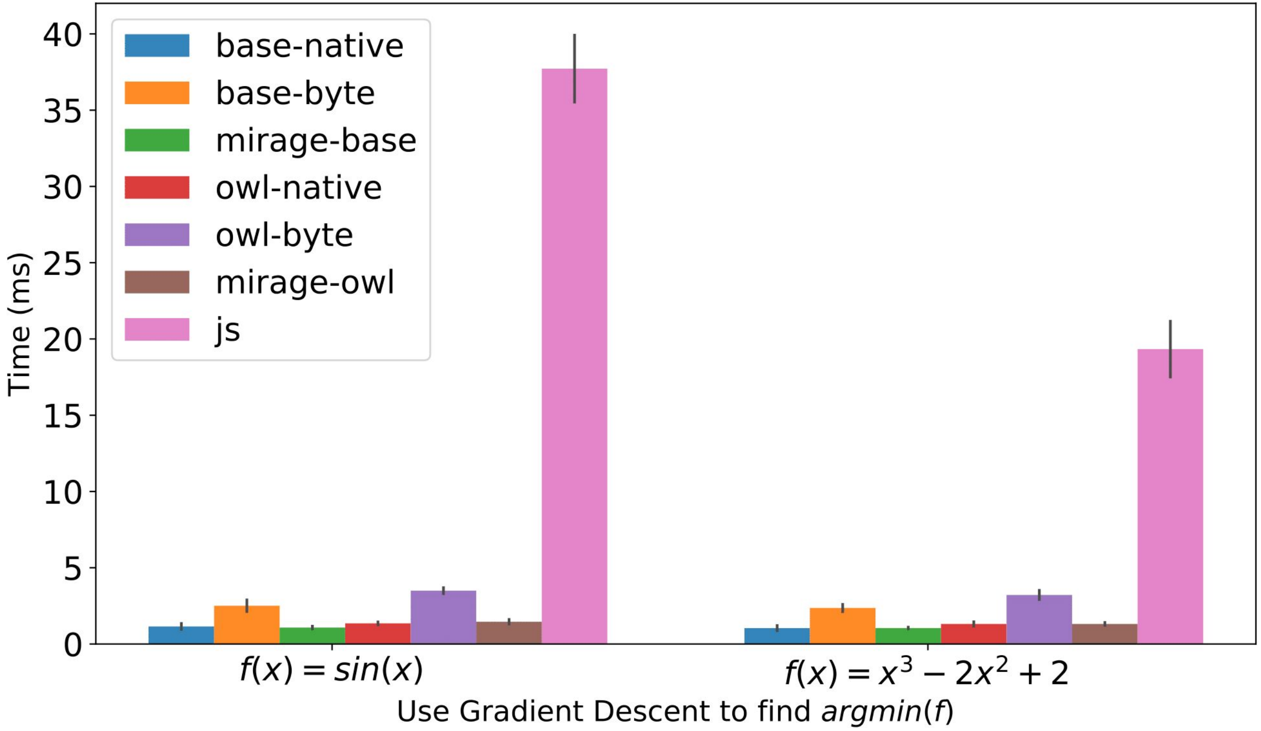
- Backends:
    - Container (Restful API)
    - Javascript
    - MirageOS

Evaluation: Backends



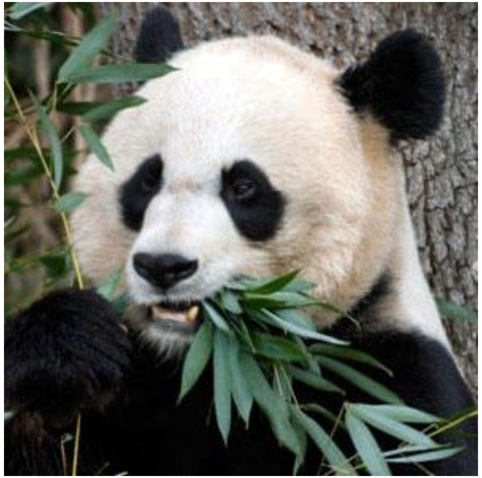


Evaluation: Backends



UNIVERSITY OF CAMBRIDGE

Application: InceptionV3



```
Top 5 Predictions:
Prediction #0 (96.20%) : giant panda,
panda, panda bear, coon bear,
Ailuropoda melanoleuca
Prediction #1 (0.12%) : lesser panda,
red panda, panda, bear cat, cat bear,
Ailurus fulgens
Prediction #2 (0.06%) : space shuttle
Prediction #3 (0.04%) : soccer ball
Prediction #4 (0.03%) : indri,
indris, Indri indri, Indri
brevicaudatus
```

- One of the most complex computer vision DNN; 1000 classification categories.
- 100 LoC for the whole network structure vs. TensorFlow’s ～500 LoC.

Gist: https://gist.github.com/jzstark/ba52dc005f135cafb4d3fbc6006291bb

Image: Panda Mei Xiang, Washington Post, goo.gl/vFmG82

Application: Fast Neural Style Transfer

Combining the content of one image with the style of another image using convolutional neural networks.

Implemented with 110 LoC.

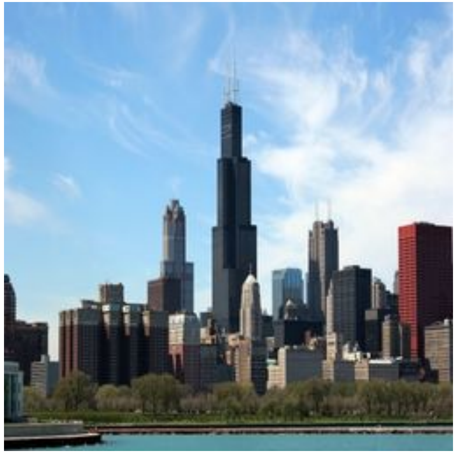
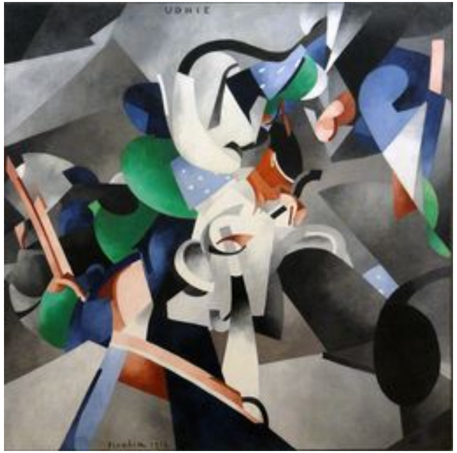
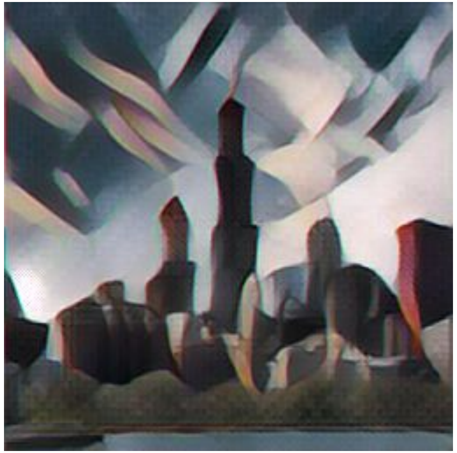


Image: Chicago view, from
www.usalifestylerealestate.com/illinois



“Young American Girl, The Dance”
by Francis Picabia



Gist: https://gist.github.com/jzstark/f937ce439c8adcaea23d42753f487299

Application: Zoo Code

Fast Style Transfer:

```
#zoo
"f937ce439c8adcaea23d42753f487299"

FST.list_styles ();; (* show all
supported styles *)
FST.run ~style:1
"path/to/content_img.png"
"path/to/output_img.jpg"
```

Image classification:

```
#zoo "9428a62a31dbea75511882ab8218076f"

let img = "/path/to/your/image.png";;
let labels = InceptionV3.infer img;;
let labels_json    = InceptionV3.to_json
~top:5 labels;;
let labels_tuples =
InceptionV3.to_tuples labels;;
```

Conclusion and Future Work

- We identify two challenges of conducting data analytics on edge: service composition and deployment.
- To address them, we propose Zoo that 1) provides a simple DSL to enable easy and type-safe composition, and 2) utilizes multiple backends to accommodate different edge deployment environment.
- We show the expressiveness of Zoo with real-world use cases, and we also evaluate performance of different backends.
- Future work: mathematical support for DSL, extend it more operations, application in networking, engineering work.

Thank you! Questions?

Email: jianxin.zhao@cl.cam.ac.uk

Owl Project: http://ocaml.xyz

UNIVERSITY OF CAMBRIDGE