Bachelor Thesis

Communication issues: Down the rabbit hole of language interoperability

by

Kai Erik Niermann

(2720905)

First supervisor:	Atze Van der Ploeg
Daily supervisor:	Atze Van der Ploeg
Second reader:	Verano Merino

October 5, 2024

Submitted in partial fulfillment of the requirements for the VU degree of Bachelor of Science in Computer Science

Communication issues: Down the rabbit hole of language interoperability

Kai Erik Niermann
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
k.e.niermann@student.vu.nl

ABSTRACT

Your Master Thesis has to be written in English and should follow the ACM style (see the Overleaf template). We also suggest a *structured abstract* following the structure below.
Context. Write this at the end
Goal. Write this at the end
Method. Write this at the end
Results. Write this at the end
Conclusions. Write this at the end

As a very rough indication, the final thesis report typically entails, excluding appendixes, between 10 and 30 pages, with an average of 15 pages. The large variation depends on many variables including the specific field, project nature, and context. We advise to ask your supervisor if you should consider which number as a reference.

1 INTRODUCTION

Its somewhat hard to properly explain the depth of the rabbit hole I have fallen into with my thesis research. It started out somewhat simple, I had this very basic idea in mind, what if you took a part of a program, and rewrote it in different programming languages and kind of like legos slotted it into place of the original program. Now this isn't by any means a new idea, I honestly just had a mild curiosity in how that would work given that I've never really wrote and executed some distinct program which was written in two different languages. In a way that idea felt somewhat odd, especially if you consider languages with completely different execution methods; how do languages even communicate with one another.

So to begin this 'little' exploration I had to first thing of a piece of software I could use to actually test this idea with. Before doing so I thought it would be important to maybe further define my idea beyond just wanting to test out my "lego theory". As said before the concept of rewriting code is by no means new, I don't believe it would be absurd by any means to suggest that most programmers at one point or another, while writing some piece of code, or even after the fact, recognized ways in which their solution can be improved. Often times improvements or iterations on the original piece of code just naturally have to occur to fit the evolving goal for a given piece of software. There are countless examples of this but going into this but we should get back to the matter at hand. So we can say that there should be some type of reason as to why we want to rewrite a specific piece of code, in our case performance would be a pretty natural choice. While most programming languages of course aim to be performant due to differences in their design and execution methods they naturally exhibit differences.

Ok, so we want to choose a piece of software, preferably one with a performance critical component, which we want to rewrite. Looking back at my previous projects one specifically came to mind, my implementation of a simple Ray Tracer in the Julia programming languages based on Peter Shirley's wonderful guide and C++ implementation. My initial goal with this project was to learn Julia using a *totally-reasonably-sized-project* as I had heard and seen that if written in a very idiomatic manner it can exhibit performance characteristics close of that to C while offering the dynamically typed simplicity of Python. Naturally when benchmarking my final implementation my results where less then desirable. To elaborate on why my program exhibited these performance characteristics it makes sense briefly go into the core of how Julia code is actually run. There is a very nice quote I found on the Julia forums a while back which I believe effectively conveys this

> *in short, it's a reasonably good static compiler but a terrible JIT*

The first response some people might have to this is confusion. Julia is a JIT compiled languages so how is its JIT terrible. Well this comes down in large part to how its JIT works, the core of this is how it preforms online partial evaluation (OPE) w.r.t. its type system. OPE w.r.t the type system in this instance refers to the process of, at runtime (online), Julia specializes (partially evaluates) methods for which it can infer the types of every local variable from the types of the arguments or any other constants; in this instance a method is said to be "type stable". Specializing type stable methods simply means it compiles methods for the set of concrete types it can infer. Practically speaking this mechanism can also be seen as a form of **monomorphizing** a set of functions for which the JIT compiler can determine the concrete types. The reason then why Julia's JIT compiler could be called "terrible" would be that it, currently at least, does not implement profile guided optimization (PGO). The lack of PGO and its reliance on type stable code for speed means that there is a causation between idiomatically written Julia and its performance. With this understanding it might now be apparent why my Ray-Tracer preformed so poorly, as I was not fully aware of this strong dependence and I simply wrote the code how I would write something in any other dynamically typed high-level language.

We've now established that we have a piece of performance critical software written in a very non-idiomatic and by extension inefficient manner. Furthermore we've seen that its execution method is not only different from ahead of time (AOT) compiled languages but also from other JIT compiled languages by its unique approach to generating machine code. The conventional route, and quite frankly recommended route, one should go here would be to simply rewrite everything with Julia's idiomatic principles in mind, though this would lead to a rather short and likely failing thesis,

so question the alternative, what if we rewrote the most performance critical segment of our ray tracer in a language which has less reliance on a unique idiomatic programming style. While at first this seems somewhat nonsensical it has the potential to raise and in turn answer some interesting questions about interoperability and more generally code execution methods including but not limited to:

- How do we design a system of communication between languages which exhibit considerable fundamental differences?
- How do we reconcile different type systems in language interoperability?
- What are the performance characteristics of language interoperability?

2 BACKGROUND

To further gain insights into this world of interoperability I thought it best to continue on with the idea I had of rewriting a component in different languages and then applying some system of interoperability. Decomposing this idea into concrete steps we can develop a rough design of how this study will be conducted.

(1) Literature review & Background
  (a) Review current landscape of language interoperability
  (b) Choose suitable libraries to explore our specific use case
  (c) Review the intersection of interoperability and program execution methods
(2) Implementation
  (a) Profile the Julia application to extract the most performance intensive piece of code
  (b) Rewrite this piece of code in different languages, preferably ones which have very different execution models
  (c) Write some type of glue layers utilizing the chosen libraries to enable this interoperability
(3) Analysis & Development
  (a) Understand how the type systems of these different languages interact, especially considering types beyond primitives
  (b) Profile the programs for all given languages to understand the performance impacts
    (i) Understand the underlying nature of the performance impacts and how they relate to differences in execution models
  (c) Understand how interop libraries are designed and enable communication between languages
    (i) Attempt to develop certain solutions for any limitations encountered

Literature review and Background

Foreign Function Interface (FFI)

FFI is a generic term which describes any implementation which enables a piece of code written in one language to call functions written in another. As the name implies these implementations usually provide some interface; an abstraction describing for example a method signature; enabling one language to call a method in another language utilizing this interface. In practice FFI usually comes in terms of an Application Binary Interface (ABI) which describes a common standard for how to compile source code into machine code and shared library providing the common code both languages which to call. Two languages written in a difference source code but able to compile to some common ABI standard have interoperability. By far the most common ABI which many languages can compile to and by extension have interoperability with is the C ABI. The point at which some source language *A* calls some target language *B* is called the FFI boundary. Some examples of this boundary in different languages can be seen in the following.

```c
// mylib.c
#include <stdio.h>

void greet(const char* name) {
    printf("Hello, %s!\n", name);
}
```

```
ccall ((: greet, "mylib"), Cvoid, (Cstring,), "World")
```

```
#[link(name = "mylib")]
extern "C" { fn greet(name: *const c_char); }

fn main() {
    let name = "World";
    let c_name = CString::new(name).expect("CString::new
        failed");
    unsafe {
        greet(c_name.as_ptr());
    }
}
```

```
mylib = ctypes.CDLL('./libmylib.so')

mylib.greet.argtypes = [ctypes.c_char_p]
mylib.greet.restype = None

name = b"World" # Use bytes for C string
mylib.greet(name)
```

One critical thing to observe is in all instance we must ensure the FFI boundary method and its arguments conform to the memory layout of the target language as specified by the ABI. Since string types for example generally have different underlying representations depending on the language here we must ensure the string has the form of a C-string, that is its a basic pointer to an array of characters ‘char*‘.

This in turn limits FFI boundary methods by the syntax and underlying ABI of our target language. For example as C has no generics, thus we cannot utilize a generic method as an FFI boundary despite generics existing in our source language an Rust being compatible with the C ABI.

```
extern "C" {
    fn add<T>(a: T, b: T) -> T;
}
```

Stated differently FFI boundary method signatures must mirror those of the target language. We can create some flexibility here by abstracting the boundary utilizing wrappers; commonly macros; which make defining the boundary closer to that of the source language while implicitly ensuring that the actual underlying boundary is valid, an example of this is the '@ccall' macro in Julia.

```
1  @ccall library.function_name(argvalue1::argtype1,
       ...) ::returntype
2  @ccall function_name(argvalue1::argtype1, ...) ::returntype
```

But this doesn't really address the underlying issue that our FFI boundary is limited by the target language.

Application Binary Interface

This is very much where the root of interoperability lies, and also its various issues. ABI stability is a key factor to understanding the current landscape of not only interoperability between languages but also within different versions of languages themselves. As the name implies the stability of an ABI refers to its specification, that being how a compiler generates machine code, being stable or unchanging. An ABI being "fully stable" implies that all aspects of its specifications will not change with any future modification of its execution system (JIT compiler, AOT compiler, etc.). One of the primary reasons for so many languages having interop with C is due to the fact that it has full ABI stability, thus there exist essentially a promise that a compiler can always generate the same machine code for a given source language that conforms to the defined C ABI standard knowing this will never some day break due to changes in C's calling conventions, memory layout, etc.

The other side of this is ABI instability, most languages currently have what could be considered partial ABI stability. What this means in practice is that most languages have some execution system which generates machine code, commonly this version or a set of versions have a common ABI. Within this set of versions there exists stability, meaning shared library calls; which are reliant on ABI stability; can/could function with any other languages capable of producing machine code which conforms to this ABI. Though across different ABI's this communication breaks down as we no longer have a homogenous standard generating the machine code but conflicting and often incompatible standards.

This nature of partial stability in the ABI's of most languages creates one of the primary limitations to language interoperability with FFI, due to the rather complex and to an extent controversial nature of having an ABI which is both fully specified and consistent across versions there quite simply are not alot of languages which have this and by extension not alot of compilers which chose to implement an ABI for this. Furthermore there is the point that if for some set of languages there is an existing C ABI already implemented, this in turn implies we have communication between these two languages with the common language taking the form of the C ABI. It should be clarified that while interop in many languages occurs with the C ABI as the basic common ground.

C-ABI and Interop in practice

To describe how interop utilizing the C ABI actually plays out in practice lets express two arbitrary languages as L_1 and L_2 . In the case of primitives the generated machine code would naturally produce matching layouts, thus little additional work is involved beyond defining the FFI boundaries in both languages. As we start getting to more complex types we start wanting to pass more complex things across the FFI boundary that might not have as straightforward of a representation in C we run into the first issue with pure C ABI based interop. This being that it leads to considerable manual specification and redefining languages specific syntax, otherwise called marshalling or boxing through the FFI boundary and in turn unmarshalling in the target language. Additionally we must ensure that the representation of our type in L_1 matches that in L_2 which for a large library could mean having to manually specific all types such that we establish a mirrored relationship; while this is somewhat obvious that it has to happen the manual nature of it would be less then desirable in the case of very large libraries we wish to interop with.

To address these issues of having to manually box everything such that it is compatible with the C ABI various language features and libraries exist as abstractions to enable more seamless and simplified interop between languages. The common features with these interoperability abstraction systems are usually automatic type (code) generation through parsing and implicit boxing and unboxing of complex types that cant be trivially passed through the FFI boundary. Examples of this are Rusts CXX library or the way Swift implemented interop as a direct part of its compiler.

Through these abstractions interop between many languages has become a mostly trivial task, in that most of the manual activities are now generally automated in one way or another, though as stated before due to the fact that this is all ultimately still working through the C-ABI (or some slight modification of it) we are still very much limited in the things we can actually pass between languages. Though these limitations are generally not large enough to really discourage the usage of FFI as the primary means of interop.

So to briefly summarize we can see that FFI generally relies on a common denominator ABI between languages; commonly the C ABI; and a set of tools to automate any manual boxing and unboxing such that the developer can - for the most part - simply define a boundary in both languages and use said boundary as if it were a regular call. An important final consideration with these FFI systems is how optimization works across language boundaries. Since the methods of shared libraries are only resolved at link time no optimizations except LTO can occur. The shared object file from the target language has optimizations applied from its own compiler then depending on if the compiler has Link Time Optimization (LTO) implemented we can optimize the whole program across the language boundary, otherwise no optimizations occurs across the language boundary.

WebAssembly (WASM)

WASM offers an interesting insights into how language interop is developing on the web. First announced in 2015 and then released in 2017 WASM is a byte-code format designed to be executed within web-browsers using stack-based Virtual Machines (VM) though has now also been implemented to run both via AOT and JIT compilers. The WASM specification was designed by the World Wide Web Consortium (W3C) but has various compiler implementations by different groups. WASM when used as a compilation target for other languages not only allows potentially arbitrary languages to run on the web, but through its ABI also allows

for another common denominator of communication. To understand the state of its ABI we have to examine the different compiler toolchains that target WASM. Currently the major ones are

What we can understand from this is that it seems like WASI will likely be the ABI to which wasi-wasm will be the likely target most compilers aim to ultimately achieve, furthermore it seems WASI seems to aim for stability with its canonical ABI, which could suggest that we see a new common standard of interoperability, especially as wasm compilers evolve. One distinguishing factor of specifically WASI's canonical ABI is development of the **component model**, the component model refers to type definitions written in WASM Interface Type (WIT), which is a interface description language (IDL) used to express types in a language independent manner. From the defined WIT code we can generate the corresponding bindings in our language thus mitigating the need to any type-shenanigan's in the case of interop since our generated bindings will by default be compatible with the ABI with no issues. It should be noted that as the documentation specifies WIT implements contracts, that is similar to how an 'extern' block expresses a function which must be ABI compatible so to do the generated bindings from the WIT code all inherently imply compatability with the underlying ABI.

Thus if you have say some library compiled to wasi-wasm which utilizes the component model to specify the API you can in turn use the library specification to generate ABI compliant bindings for your source code, then use a wasi-wasm compiler and have full cross-language interoperability without the need for any complex wrapper libraries. Though once again here we are limited by the expressiveness of the component model and corresponding ABI, meaning that if a library wants to make something available for use externally it has to conform to the component model thus potentially limiting certain language specific features for a cross language setting. Unless similarly to the C ABI we adjust the compiler to support these language specific features and in turn break compatibility with the ABI.

3 RESEARCH METHODS

3.1 Ease of use and Performance

4 IMPLEMENTATION

4.1 Python Component

4.2 C++ Component

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

5 DISCUSSION

Here you put your results in context (possibly grouped by research question). Usually, this section focuses on analyzing the implications of the proposed work for current and future research and for practitioners.

Compiler/Toolchain	Description
Emscripten	This was the first project to define a POSIX compliant ABI for WASM, though as of 2020 at least it is not stable.
Cheerp	Seem to support WASI-WASM as a target for the compiler. Inherently has interop with JS because it literally compiles to JS uses ts2cpp for creating C++ interfaces from TS types.
WASI	A subgroup of the WASM community group which is aiming to define a set of APIs and a canonical WASM ABI, currently the actual implementation of this specification comes in the form of: • Wasienv which can compile Swift and C++ to wasm conforming to the WASI standard. • wasi-sdk which seems to be some more low level compiler for wasi, i think the like backend, so you just need to implement the frontend for specific languages. • wasmtime another wasm runtime which also has WASI as a target specification it seems. • wasix compatibility layer to make WASI POSIX compliant when built with the WASI SDK. • wasm32-wasi seems to be a frontend for Rust again using WASI. WASI with the component model seems to have the most thoroughly documented ABI.
GoJS	Not much info on this, just seems that a go compiler has wasm as a target.

Table 1: Major WASM Compiler Toolchains and Their Descriptions

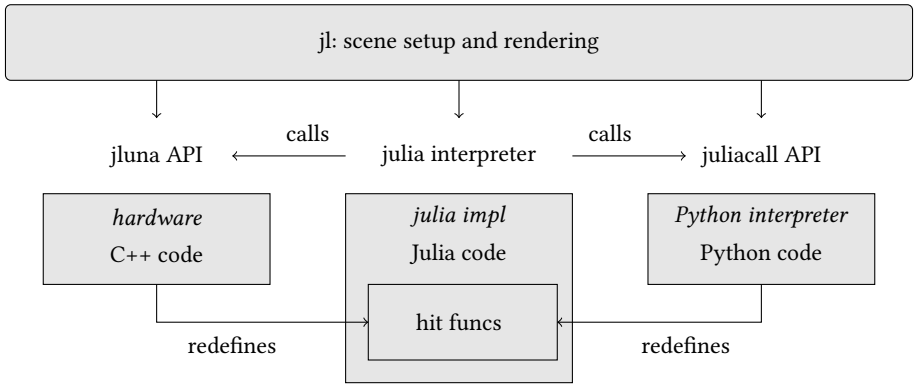

Figure 1: *System architecture of multi-language PBR renderer.*

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim.

Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetuer at, consectetuer sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui.

6 RELATED WORK

Describe here scientific papers similar to your thesis work, both in terms of goal and methodology. One paragraph for each paper (we expect about 5-8 papers to be discussed). Each paragraph contains: (i) a brief description of the related paper and (ii) a black-on-white description about how your work differs from, or overlaps with, the related paper, hence emphasizing the novelty contributed by this thesis. You may place this section immediately after the Background section, if necessary.

Sed lobortis, justo et pretium lobortis, mauris turpis condimentum augue, nec ultricies nibh arcu pretium enim. Nunc purus neque, placerat id, imperdiet sed, pellentesque nec, nisl. Vestibulum imperdiet neque non sem accumsan laoreet. In hac habitasse platea dictumst. Etiam condimentum facilisis libero.

7 CONCLUSION

Briefly summarize your contributions, and share a glimpse of the implications of this work for future research.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.