*1. Summarize the (at most) 3 key main ideas.*

The 3 key ideas presented in this paper is the program representation, the architecture, and subsequent experimental results of LLVM.

1. The program representation of the compiler makes use of static single assignment, language-independent type (yet not type-safe) system, explicit access memory model, high level exception handling representation in low level code, and unified offline and compiler representation.
2. This format provides high-level information in a low-level form that allows the compiler architecture to fully optimize at many levels—from compile-time, linker and interprocedural, and offline code generation optimizations, to run time path profiling and user profiling reoptimizations.
3. As a result, LLVM was experimentally tested on C/C++ and yielded better type information for aggressive optimization in a comparably equal or smaller sized machine code than current native machine code and in a much shorter time compared to tradition compilers with optimization.

*2. State the main contribution of the paper.*

The main contribution of this paper is that it describes a workable and experimentally proven compiler framework that comprises of 5 distinct capabilities that offer a more complete and lifelong performance of a program that was not previously obtain by other compilers– (1) Persistent program information, (2) Offline code generation, (3) User-based profiling and optimization, (4) Transparent runtime model, and (5) Uniform, whole-program compilation.

*3. State the limitation of the paper.*

The limitation of this paper is that it was only capable, at the moment, of experimentally representing and compiling C/C++ programs and therefore can only theoretically assume the benefits of other high level languages.

*4. Find at least one open question and try to answer it. One example open question for the LLVM paper is: What's the difference between LLVM and GCC?*

Has the size inefficiency caused by SSA, requiring 64-bit or larger register encodings, been resolved since the paper has been written?

I could not find specific acknowledgement to maintain a 32-bit encoding for SSA throughout a program, but there have been papers written to make SSA more efficient in general such as “Increasing the scope and resolution of Interprocedural Static Single Assignment” by Silvian Calman and Jianwen Zhu and “Formal Verification of SSA Optimizations for LLVM” by Jianzhou Zhao, Santosh Nagarakatte, Milo M K Martin, and Steve Zdancewic.