We thank the reviewers for their insightful comments. We acknowledge the feedback about improving the writing and adding details; we promise to incorporate it in the final version.

**Comparison against previous compilers (Reviewer D):**

The comparison is in Section 2.1. To summarize, CBMC-GC times out after 5 hours when multiplying two tiny matrices; ObliVM scales better but is still 25x slower than EzPC on the same hardware.

**Novelty of EzPC (Reviewer D)**

The novelty of EzPC is in implementing cryptographic cost awareness inside a compiler, allowing the programmers to program in a high-level language absent of all the cryptographic details, while the compiler chooses the best circuit representation (boolean or arithmetic) for different parts of the program and also handles interconversions between them -- all in a provably secure manner. More specifically:

**w.r.t previous compilers:** Wysteria, ObliVM, and CBMC-GC, use boolean (or arithmetic) circuits only backends. And so, the challenges associated with multiple representations and interconversions are absent. Moreover, as shown in Section 2.1, they perform poorly on ML tasks that involve a mix of arithmetic and boolean computations.

**w.r.t. [34]:**  [34] operates over circuits represented as a sequence of dyadic operations, while EzPC provides a high-level programming language with loops and branches. EzPC compilation is automatic (Section 3) and sub-quadratic in the size of generated ABY programs (e.g. Figure 3) after switching off memory safety checking. In sharp contrast, [34] describes approaches that would be doubly exponential and exponential in the length of ABY programs.

**w.r.t Yao pipelining:** Unlike prior work, which is specific to and works on Yao’s garbled circuits, our partitioning technique works at the source level and is agnostic to the cryptographic backend.

These features (ease-of-programming, use of multiple crypto backends, and scalability through partitioning) enable us to write and evaluate a wide array of ML algorithms such as matrix factorization with efficient generic 2PC techniques. Without EzPC, this would have been a near-impossible task and has never been done.

**Scalability experiments for Partitioning (Reviewer D):**

These are in Section 7, Figure 11. For DNNs with 6 layers or more, swap causes slowdown and large programs terminate with SIGBUS.

**Loops, branches, and secret array indices (Reviewer A and B)**

In our experiments, assigning public labels to control flow variables is critical for performance and/or security. However, we have observed that conservatively creating circuits for other public variables incurs minimal cost in practice.

At the expression level, EzPC provides secret conditionals (using the ‘? :’ operator) which are implemented using multiplexers. Secret dependent array accesses can also be implemented using the ‘? :’ operator.

**Other comments:**

**Reviewer A:**

Partitioning is fully automated and compatible with any 2PC protocol. The EzPC compiler, with suitable changes to operator costs, can target other backends including 3PC or MPC, or protocols secure against malicious adversaries. We are currently pursuing these directions.

The proofs follow by straightforward induction once the key lemmas (Appendix C) have been identified. We will include full proofs in a Techreport.

We have evaluated against handwritten ABY code for the examples in the ABY repository and EzPC has comparable performance. However, these examples are not representative of practical ML tasks. Writing sophisticated ML applications such as neural networks in ABY is a very tedious and nearly impossible engineering task for non-crypto experts -- automating the same is our main contribution.

**Reviewer B:**

Scalability issues: when the circuit size is larger than the memory size, it either runs slowly or not at all (Figure 11). Partitioning enables decomposing g1;g2 that does not fit in memory into g1 and g2, both of which are individually small enough to fit in memory.

**Reviewer C:**

We thank the reviewer for providing us with references to additional interesting functionalities that could be written using EzPC.