Rigorous Specification and Low-Latency Implementation of Technical Market Indicators

Rigorous Specification and Low-Latency Implementation of Technical Market Indicators

Konstantin Bakanov  Ivor Spence  Hans Vandierendonck
Queen’s University of Belfast
kbakanov01@qub.ac.uk  i.spence@qub.ac.uk  h.vandierendonck@qub.ac.uk

Abstract
Technical market indicators are tools used by technical analysts to understand trends in trading markets. Technical (market) indicators are often calculated in real-time, as trading progresses. This paper presents a mathematically-founded framework for calculating technical indicators. Our framework consists of a domain specific language for the unambiguous specification of technical indicators, and a runtime system based on Click, for computing the indicators. We argue that our solution enhances the ease of programming due to aligning our domain-specific language to the mathematical description of technical indicators, and that it enables executing programs in kernel space for decreased latency, without exposing the system to users’ programming errors.

Keywords  domain-specific languages, technical market indicators, in-kernel processing, click router

1. Introduction
Technical Analysis is concerned with analyzing stock price movements, identifying patterns in them and then taking advantage of these patterns by making preemptive trades [12]. Traders predominantly relying on technical analysis to conduct their trading are sometimes called “technicians” or “technical analysts”. There are many tools which technicians use to first describe and later identify the patterns in stock movements. These range from charting, where traders visually identify patterns in charts, technical market indicators, which usually consist of some mathematical formula encompassing a sliding window view of the market, to more complex tools, such as neural networks or a combination of neural networks and technical market indicators [20, 21].

A technical market indicator is a mathematical formula applied to time series data, which usually consists of high, low, close prices and volume for a given period of time. Conventionally this period of time used to be a trading day, however in the modern age with the rise of day traders, the period of time can be anything from a few seconds to a few hours [14, 23]. The technical indicators are therefore used to identify trends within markets and their associated characteristics, such as momentum, sentiment and other properties [8]. Some indicators are used to gauge an entire market, such as Coppock Curve and Advance-Decline Line.

A system to compute technical market indicators in real-time has to combine various properties. It must allow technicians to fairly easily specify the indicators, the specification of the indicators must be rigorous and they must be executed as fast as possible. These are significant challenges, but ones that are well addressed by domain specific languages (DSLs). In this paper, we develop the basis for a domain specific language for specifying and computing technical market indicators.

The specification of technical market indicators is non-trivial. Figure 1 lists the Commodity Channel Index, a price momentum indicator. Clearly, such a specification is not only hard to read, but it is also imprecise. To counter these issues, the specification comes with a detailed computational recipe [7], resembling an algorithm written in an imperative programming style. We have found that there is generally no uniformity of notations. This may lead to ambiguity in interpreting indicators as our discussion on the definition of
time intervals will show. This work aims to solve these issues by presenting a mathematically founded framework for the
definition of technical indicators.

In order to make the specification of technical indicators rigorous, we base our DSL on the mathematical specification
of technical market indicators, as elaborated in Section 2. In essence, a technical indicator operates over a sequence
of timestamped values (trades), which are summarized into high, low, and close prices over specific time intervals.
Technical indicators are then functions over sequences of high, low, open and close prices. Our DSL will be closely
aligned to these mathematical definitions, closing the semantic gap between the specification (mathematics) and the
programming language.1 Moreover, our formulation is modular, as different operations may be defined individually and
reused at will. As an example, we define the exponential weighted moving average (EWMA) as a reusable compo-
nent in Section 3. This way, we can build up a library of commonly used computations in technical market indicators,
which simplifies the programming task.

We apply our framework to the formulation of the direc-
tional movement index (DMI), a non-trivial technical market
indicator [9]. Section 4 shows how we can specify DMI in a
few steps by abusing definitions of high, low, open and close
prices as well as EWMA.

We select the Click modular router [15] as a runtime sys-
tem for our DSL. As such, it is invisible to the programmer
and other runtime systems may be selected or designed.
Click is, however, particularly appropriate as it is designed
to react to incoming (network) packets (in our case trades)
and process packets as they arrive. The mapping of func-
tional components in our DSL to Click components is quite
straightforward and is well suited to apply code optimization
strategies to specialize the Click components to the technical
indicators computed. Moreover, the Click router is capable
of working with raw market data, coming directly from the
exchange (known as a market data feed), thus making it pos-
sible to use it as part of a low latency system, which can be
appealing to high frequency traders.

As the latency of calculating technical market indica-
tors can be extremely important in algorithmic trading, we
demonstrate the use of our DSL to perform the computa-
tion of indicators in kernel space (Section 5). Executing pro-
grams in kernel space is extremely dangerous, as any soft-
ware bug may affect the whole system. However, by using a
DSL, where the programming language is restricted and
where memory accesses are under full control of the DSL
compiler and runtime system, we can provide a safe system
to execute programs in kernel space. This way, we demon-
strate improved processing latency/throughput for the DMI
indicator when evaluating the indicator in kernel space.

Finally, Section 6 concludes this paper and discusses fu-
ture work.

2. The Fundamentals of Technical Indicators

The basic building blocks of a technical indicator are the
following properties: High Price, Low Price, Closing Price,
Opening Price (rarely) and Volume. Any one of these char-
acteristics is calculated for a given time interval. Once those
values are calculated for every interval in question, then any
technical indicator can be represented as a rigorous mathe-
matical formula.

Some indicators are simple and can be recorded with one
expression, as in the previous CCI example. Some formulas
are more complex and are better recorded as a cascade of
functions.

However before moving onto the more complex con-
struts let us consider first the most basic building block of
any technical indicator - that is the time-price interval. For
this purpose let us define the prices, the volumes and the
times at which these trades have occurred as a sequence of
3-tuples. Let us name this sequence $D$. In order to select any
one member of a given tuple $d$, let us define 3 helper func-
tions:

- $time(d)$ - which returns the time component of a tuple $d$.
- $price(d)$ - which returns the price component of a tuple $d$.
- $volume(d)$ - which returns the volume component of a tuple $d$.

Let us also assume that the observations started at time 0.

Now we need to select values $d$ from the sequence $D$, which fall inside a given interval. We need this to calculate
High, Low, Close, Open, as these are defined per a given
interval. However we have a number of choices in this.
Firstly let us name the length of interval $a$, let us name the
current time $T$ and let us number the intervals. We’ll use
variable $i$ for this purpose, so that the first interval would
be $i_0$, the second interval would be $i_1$, the one after $i_2$ and
so on. The number of current interval $k$ can be calculated
as $k = \lceil T/a \rceil$. Then the main question can be formulated
as this: how do we select values $d$ (for calculating High,
Low, Close, Open) when time $T$ occurs inside an interval
of $i_k$? There are 3 options here:

1. We only calculate High, Low, Close, Open after an inter-
val has elapsed. I.e. during an interval $i_k$ the most recent
High, Low, Close, Open would be the ones calculated for
the previous interval $i_{k−1}$. Therefore if a technical indica-
tor operates on High, Low, Close, Open from, say, 3 last

```
Figure 2. Selection of time intervals under options 1 and 2. Intervals are fixed and are all of equal length.
```

\[ i_0 \quad i_1 \quad i_2 \quad \ldots \quad i_k \]

1 We do not yet define a DSL in this paper, as we are still exploring the
principles and appropriate concepts to use, yet the mathematical description
and DSL are in direct correspondence.
Let us start with defining a generic function \( F \), which does the selection, and applies an arbitrary function \( X \) to the subsequence of values \( d \) that fall within a selected interval.

For the option #1 the formulas will look as follows:

\[
F_1(D, T, i, a) = \{d \in D \mid (k - i) \ast a \leq \text{time}(d) < (k - i + 1) \ast a \land (k - i) \geq 0\}
\]

where
\[
d \text{ is a dummy variable.}
\]
\[
T \text{ is the current time.}
\]
\[
a \text{ is the length of the interval in question.}
\]
\[
k \text{ is the number of the current interval, calculated as } k = \lceil T/a \rceil.
\]

\( i \) is the offset of the interval in question from the current interval. By definition \( i \) is equal to or greater than 1.

This formula reads like this: for every \( d \) in the sequence \( D \) select the ones, whose time values are greater than or equal to \((k - i) \ast a\), but less than \((k - i + 1) \ast a\). \((k - i) \geq 0\) is used to ensure that we select a valid interval.

For the option #2 the formula 1 will hold for the intervals \( i_{k-1} \) and below, but for the interval \( i_k \) it will change, so that:

\[
F_2(D, T, i, a) = \begin{cases} 
if (i > 0) \text{ then } F_1_1 \\
else d : D | T - \Delta t \leq \text{time}(d) \leq T \end{cases}
\]

where \( \Delta t \) is the time between \( T \) and the end of the previous interval. It can be computed as \( T \mod a \).

For the option #3 for all the intervals the formula will change to:

\[
F_3(D, T, i, a) = \{d : D | (\Delta t + (k - i - 1) \ast a \leq \text{time}(d) < (k - i + 1) \ast a \land (k - i - 1) \geq 0)\}
\]

Each of \( F_{1-3} \) would be preferred for different use cases. E.g. \( F_1 \) would be preferred for a case, when a user does not mind if an indicator is slightly lagging (up to the length \( a \) of an interval) behind the current state of the market. \( F_2 \) would be preferred when a user is interested in a more up-to-date state of the market than what \( F_1 \) provides. \( F_3 \) would give the most accurate look of the market, but would require more computational resources.

In practice, option #2 is the most common [10].

Using function \( F_2(D, T, i, a) \) we can now define functions that help us calculate High, Low, Open, Close prices for the interval selected. Also it is important to understand that sometimes for rarely traded instruments we may receive no trades in a given time interval. We handle this situation by copying the High/Low/Open/Close price from the previous interval. If the previous interval has no prices, we continue traversing our sequence \( D \) until we reach the start of our observation at time 0 in which case the resultant value is 0.

Let us start with defining a generic function \( I \), which searches for non-empty interval \( i \) and applies an arbitrary function \( X \) to the subsequence of values \( d \), that fall within a selected interval.
current interval’s volume is 0.
retrieve the volume from any of the previous intervals if the
interval. Unlike High/Low/Open/Close we do not need to

functions for computing High, Low, Open and Close
can then be defined as follows:

where
ε is an empty sequence.
X is an arbitrary function that takes a sequence as
a parameter and returns an element of that sequence
(e.g. head, which returns the first element).

The functions for computing High, Low, Open and Close

High:

where
maxp is the function returning an element of a se-
quence of 3-tuples with the highest price.

Low:

where
minp is the function returning an element of a se-
quence of 3-tuples with the lowest price.

Close:

where
last is the function returning the last element of a
sequence.

Open:

where
head is the function returning the first element of a
sequence.

Some indicators take into account the volume for a given
interval. Unlike High/Low/Open/Close we do not need to
retrieve the volume from any of the previous intervals if the
current interval’s volume is 0.

vol(D,T,i,a) = \sum_{j=0}^{#F_2(D,T,i,a)-1} volume(F_2(D,T,i,a),j)

where
j is used to select a member of a sequence, returned by F_2 function [11].
volume is used to select the volume member of a 3-
tuple as defined earlier in this section.
#F_2(D,T,i,a) is used to retrieve the length of a
sequence.

3. EWMA - Exponentially Weighted Moving
Average

Many indicators (including DMI indicator, which we discuss
in the following sections) use EWMA. For the benefits of
performance usually an EWMA is calculated according to
this formula [19]:

\[ \hat{X}_{0,\alpha} = X_0 \]
\[ \hat{X}_{k,\alpha} = (1 - \alpha) \ast X_k + \alpha \ast \hat{X}_{k-1,\alpha} \] (10)

where
\( \hat{X}_k \) is the value of exponentially smoothed price for
the interval k.
\( X_k \) is the price for the interval k.
\( \alpha \) is a weighting parameter, which is chosen by the
user. It needs to be in the range 0 < \( \alpha \leq 1 \).

From this formula one can see that we only need to store 1
previous value of EWMA to calculate the present value of
EWMA.

Let us re-write the above formula for EWMA using our
notation. First we need to calculate EWMA for the first
period:

\[ EWMA_X(D,T,k,a,\alpha) = X(D,T,k,a) \] (11)

where
k is the number of the current interval, calculated as
k = \( \lfloor T/a \rfloor \).

By using k as a value for i we can refer to the first (in-
dex 0) interval of our observations. Thus in the above
formula we set the EWMA value of a first interval to
the value of function X of that same interval.

Recall the meanings of the other variables:
D is the sequence of 3-tuples, denoting time, price
and volume of the trade.
T is the current time.
\( a \) is the length of an interval.
\( X(D,T,i,a) \) (with k as a value for i) is used to
denote \( H(D,T,i,a), L(D,T,i,a), C(D,T,i,a), \)
\( O(D,T,i,a) \) or any other functions derived from
these.

For every subsequent interval EWMA looks like follows.

\[ EWMA_X(D,T,i,a,\alpha) = (1 - \alpha) \ast X(D,T,i,a) + \]
\[ + \alpha \ast EWMA_X(D,T,i+1,a,\alpha) \] (12)

where
i is the offset from the current interval.

4. DMI indicator

Let us now demonstrate our mathematical concepts in action
by creating a rigorous model of Directional Movement Index
(DMI) indicator. DMI is characterized as “a rather complex
trend-following indicator” [9].
First, positive and negative directional movements are calculated - named PDM and NDM respectively. PDM is current period’s high minus previous period’s high. NDM is previous period’s low minus current period’s low.

\[
PDM_{\text{tran}}(D,T,i,a) = H(D,T,i,a) - H(D,T,i+1,a)
\]

(13)

\[
NDM_{\text{tran}}(D,T,i,a) = L(D,T,i+1,a) - L(D,T,i,a)
\]

(14)

where

\(i\) is an offset from current interval \(k\), which is computed as \(k = \lceil T/a \rceil\).

\(\text{tran}\) stands for transitional and denotes transitional PDM and NDM respectively, i.e. not the final ones.

PDM is zero if \(PDM_{\text{tran}}\) is negative, or less than \(NDM_{\text{tran}}\). Otherwise it is equal to \(PDM_{\text{tran}}\). The same logic applies to NDM.

\[
PDM(D,T,i,a) = \text{if } (\text{PDM}_{\text{tran}}(D,T,i,a) < \text{NDM}_{\text{tran}}(D,T,i,a)) \text{ then } 0
\]

\[
\text{else max}(\text{PDM}_{\text{tran}}(D,T,i,a),0)
\]

(15)

Likewise NDM is:

\[
NDM(D,T,i,a) = \text{if } (\text{NDM}_{\text{tran}}(D,T,i,a) < \text{PDM}_{\text{tran}}(D,T,i,a)) \text{ then } 0
\]

\[
\text{else max}(\text{NDM}_{\text{tran}}(D,T,i,a),0)
\]

(16)

At the same time true range is calculated, which is the largest value of: current high minus current low, current high minus previous close or previous close minus current low.

\[
TR(D,T,i,a) = \max(H(D,T,i,a) - L(D,T,i,a),
H(D,T,i,a) - C(D,T,i+1,a),
C(D,T,i+1,a) - L(D,T,i,a))
\]

(17)

Then PDM, NDM and TR are smoothed with exponentially weighted moving average (EWMA).

Having smoothed PDM, NDM and TR we can now calculate positive and negative directional indices (PDI and NDI respectively), which are calculated by division of smoothed PDM by smoothed TR for PDI and by division of smoothed NDM by smoothed TR for NDI. The author of the indicator suggests that \(\alpha\) should be set to 1/14 or 0.07143.

\[
PDI(D,T,i,a) = \text{EWMA}_{PDM}(D,T,i,a,1/14)/\text{EWMA}_{TR}(D,T,i,a,1/14)
\]

(18)

\[
NDI(D,T,i,a) = \text{EWMA}_{NDM}(D,T,i,a,1/14)/\text{EWMA}_{TR}(D,T,i,a,1/14)
\]

(19)

Next directional movement (DX) is defined as the the absolute difference between PDI and NDI, divided by the sum of PDI and NDI and the result multiplied by 100.

\[
DX(D,T,i,a) =
100 \ast \text{abs}(\text{PDI}(D,T,i,a) - \text{NDI}(D,T,i,a)) / \left(\text{PDI}(D,T,i,a) + \text{NDI}(D,T,i,a)\right)
\]

(20)

where \(\text{abs}\) is the function returning an absolute value of a number.

Finally average directional movement (ADX) is defined as an EWMA of DX with \(\alpha\) set to 1/14:

\[
ADX(D,T,i,a) = \text{EWMA}_{DX}(D,T,i,a,1/14)
\]

(21)

5. Implementation

5.1 Click Platform

We have implemented DMI indicator described above on top of Click router. The Click Modular Router project [5, 16–18] takes the approach of designing small objects which can be joined together to create a workflow or configuration, in an essentially arbitrary manner. Each object, known as a Click element, implements a processing function on a packet header at one of the layers of the OSI stack. The creation of Click configuration is facilitated by the Click language, that is a domain specific (DSL) language for building routing functions. Click router can run in kernel space as well as in user space, which should result in increased performance. From a development point of view an added benefit of Click router is that despite the integration with Linux kernel as a module it still allows code to be written in C++. Click router encompasses an entire framework, which provides certain constructs, typically present in STL, but not available to kernel developer, such as hashmaps, strings, vectors etc.

- Figure 4. DMI indicator - topology.
Another advantage is Click’s multithreaded design (not used in our present tests), which allows to scale when deployed on many core CPUs.

The choice of Click platform for implementation is not binding. It can be any platform corresponding to a number of requirements. The next section expands a little on what makes Click a relatively good choice for this particular case.

5.2 Mapping Maths to Click

Our rigorous mathematical definition of a DMI indicator is essentially a cascade of interrelated functions, operating on a sequence of data (for most part). Click router has been designed specifically to process streams of data using a cascade of processing nodes (elements), which can be easily configured with the use of a DSL. Therefore the purpose of our experiments is to verify how well we can map the elements of our DSL onto the underlying platform (Click in this case, but it can be any suitable platform). We do the mapping manually. First we develop a set of Click elements (processing nodes), which correspond to functional constructs in our mathematical DSL. Second we link those elements together with the use of Click language to form a complete processing workflow - a DMI indicator. An overall picture of processing nodes and links/edges is shown in figure 4.

After processing the raw market data in chix_trade_handler (to generate trades) and selecting the symbols of interest in symbol_filter we pass the trades to direct_move, which calculates PDM and NDM, and to true_range, which calculates TR. The output of these elements is smoothed using various instances of EwmaElement. di_minus and di_plus calculate NDI and PDI correspondingly. Together, the output of di_minus and di_plus is used to calculate DX (in dx element). The output of dx is again smoothed using ewma_dx, which makes it ADX.

5.3 Experimental Methodology

Fidessa, the company sponsoring this paper, has provided the file with market data feed messages from Chi-X Europe stock exchange. This file contains about 1 hour worth of trading messages. The format of the messages corresponds to “CHIXMD FEED SPECIFICATION” [6]. In our experiments for simplicity purposes we have changed the transportation method of market data messages from TCP (as in original specification) to UDP, which is also widely used by very similar protocols, such as Multicast PITCH [1].

Since Linux kernel does not immediately allow to do floating point calculations, we had to use fixed point arithmetic library to handle prices and other floating point calculations [2].

Click version 2.0.1 is used. Click package has been compiled for kernel space execution (as a kernel module) and for user space execution (as a standalone application). Linux kernel version 2.6.34.13 has been changed to enable the interception of packets. Two hooks were added to kernel net-if_receive_skb function. The first hook is used by ti-

mestamper kernel module, which attaches a time stamp (produced by rdtsc instruction) to a network packet of interest. The second hook is used only by Click in kernel mode to intercept the packet. Intel(R) Core(TM) i7 CPU 930 processor running at 2.8 GHz was used for running the DMI indicator. This processor possesses an invariant time stamp counter [13], which allows to use it for accurate and predictable measurements. The latency was measured between the time stamp, produced by the timestamper kernel module, and the time stamp produced by StatPrinter element.

The rest of system settings were set to their default values as per standard Ubuntu distribution 12.04 and per generic Linux kernel version 2.6.34.13.

When Click is running in user space, it runs as a standalone application and simply consumes packets from a raw socket.

5.4 Results

A simple application was written, which sends UDP packets at a configurable message rate over computer network. Latency measurements were performed at 4 various rates: 10000, 50000, 100000 and 200000 messages per second. The results are shown in figure 5. SymbolFilter element was configured to filter trade messages for 2 of the most frequently traded symbols: BARG1 and LLOY1. The latency measured is an average (mean) latency calculated over 6500 updates that reached StatPrinter. Standard deviation is calculated over the same 6500 messages as well. It is important to note that only around 2% of all market data contained in a file are trade messages.

The graphs present the results as candlesticks: the line in the middle is the average (mean) latency, the top and the bottom of the box are offset 0.5 of standard deviation from the mean. The maximum and the minimum values are represented by the top and bottom whiskerbars respectively. As one may see from the graphs, the in-kernel processing latency is significantly lower than that in userspace. We believe this is in part due to short processing path of DMI indicator relative to the processing path of network stack, which we bypass. The standard deviation in kernel space is also lower, meaning that much of the variability in execution time is caused by the network stack and OS/user space context switch. At 100000 messages per second the load on the system starts being excessive, leading to higher standard deviation. At 200000 messages per second the user level version of DMI indicator cannot keep up with data rates and only circa 2100 updates are processed, the rest are dropped. However the kernel space version of DMI indicator still manages to process all 6500 updates although at a cost of increased latency.

6. Conclusion and Future work

In this paper we have outlined the underpinning mathematical concepts for the rigorous construction of technical mar-
kernel indicators. These concepts are important because in a current environment there is no common notation for technical market indicators, neither there is any uniformity of notations. We hope that our paper will help to address both of these issues. Using mathematical concepts we have produced the model of a sample indicator: DMI indicator. We then produced an implementation of our sample indicator on top of Click router. The sequence data structure used in our model has mapped well onto the stream processing nature of Click router and the functional elements of DMI indicator have mapped well onto Click’s concept of elements.

As expected, the latency of DMI indicator executing in kernel space is significantly lower than that in user space. In the future we intend to expand our work to encompass running simultaneously a number of technical market indicators in a multi-threaded environment. We will concentrate on jointly optimising the Click routing functions for a set of indicators and instruments. We will also consider the processing of historical data as a computational problem, that allows a different set of code optimisations compared to real-time processing.

7. Related Work

Various online brokerage companies provide the software platforms for trading. These platforms often include the programming IDE and the language to allow users to create their own custom indicators. Sometimes the language in question is a general purpose imperative language, such as JavaScript in eSignal, C# in Wealth-Lab, VB.NET in Stockfinder and others, but often it’s a custom-made domain specific language (DSL), such as imperative EasyLanguage in TradeStation and Multicharts, the imperative AIQ EDS language in AIQ, imperative QScript in Wave59 and others. These DSLs are usually quite expressive and once again aimed at applying mathematical formulas to time series data. Quite often they are also tightly integrated with the trading platform, allowing the coding of execution strategies, alerts and other custom objects. The differences between DSLs make migration from one platform to another not as easy and painless as it could be. One exception to this is EasyLanguage, which is used by both the TradeStation and MultiCharts.

FFTI [25] is an attempt to define a uniform notation for expressing technical market indicators. The authors take an approach whereby they define a number of aggregate functions, such as average, sum, product, min, max etc, and then use those functions with auxiliary parameters to define technical market indicators. They implement their concept as a web based tool. FFTI has little in common with our work as it uses a different approach, doesn’t provide the same low level of underpinning mathematical concepts and does not intend to be modular and stream based in a way that our concept does.

Two books [3, 7] provide some of the most comprehensive overviews on the topic of technical market indicators, including their description, returns analysis and trading advice.

From time to time kernel space execution also receives its share of attention. Shukla et al [22] evaluate the performance of user space µserver and kernel space server TUX. Birch [4] evaluates the performance of an in-kernel packet capturing utility and Zander et al [24] compare the performance of an in-kernel UDP traffic generator and receiver vs. conventional tools such as tcpdump and CRUDE. All of them conclude that in certain (if not most) situations kernel space execution is superior to user space execution in terms of latency, performance and CPU load.

Acknowledgments

This work has been carried within the Northern Ireland Capital Markets Collaborative Network funded by a consortium of Queen’s University Belfast, University of Ulster, Fidessa, NYSE Euronext, Citi, First Derivatives, Kofax and Invest Northern Ireland. The authors are grateful to Fidessa UK Ltd for providing the Chi-X data set which was used to conduct the experiments reported in this paper and for many useful
discussions with Fidessa on this work. Konstantin Bakanov is supported by a PhD studentship from the Department of Employment and Learning NI. The research was carried out at the ECIT Institute at Queen’s University Belfast.

The research leading to these results has received funding from the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme (FP7/2007-2013) under the project NovoSoft, REA grant agreement no 327744.

References