The combination of Dart, Flutter, and tflite for this application is a very smart and efficient way to go about its development. The main necessities for a project like this are support for a platform for object recognition, UI development for the app, and efficient networking capabilities to compare cost to other items online. Object recognition will be accomplished by running image data from the panoramic photo taken by the user, through tflite's machine learning algorithm using a trained neural network. UI development will be achieved through Flutter's design format of the Dart language. Finally networking will be achieved through Dart's support for asynchronous I/O and event driven programming.

TensorFlow Lite is a good choice to use for the necessary machine learning algorithm because it is a well developed platform and well supported. This means the developers will likely not need to make many changes as updates come out as long as tflite continues supporting existing APIs. It has a very strong ML support for processing neural networks which is necessary to help the app run efficiently in its native environment and quickly identify objects in the photo taken. This is helped significantly by the addition of AI accelerators in many modern mobile devices. This piece of hardware is optimized for processing AI data like neural networks very quickly by focusing on performing many rapid low cost operations. Tflite can take advantage of this piece of hardware to improve app performance. TensorFlow Lite provides direct support for some larger programming languages such as C++, Java, and Python. Although it doesn't have one for Dart, the tflite plugin serves the necessary function, but the plugin developer must maintain it with changes to Dart or TensorFlow Lite.

Flutter is a very nice SDK to use to develop this Dart application as it is designed around creating solid UIs for primarily natively run applications to optimize performance. It builds its UI around the idea of widgets which represent every bit of information displayed whether it's a button, image, text, etc. These widgets can be combined or arranged to function under another widget, like if a button is pressed it will display another screen containing a different set of available widgets. Widgets can be either stateful meaning when an internal state is updated, the entire widget will update accordingly, or stateless which will never update. This is an easy way to organize many essential app functions with one easy to use interface. This could be applied to the GarageGarner app with perhaps a button widget to take a picture of the garage sale, then go into a tflite processing function and arrange the received objects into individual widgets which could then be clicked on to compare prices.

Flutter is a helpful app-designing SDK for Dart that many of the other proposed languages do not have. Many languages have libraries to support UI development, but Flutter is built with the intention of taking Dart code and rendering it into IOS or Android applications through its Material and Cupertino libraries. It actively facilitates the transition from code to application. Java and OCaml on the other hand require other third party support for mobile app development which don't have the same level of support or optimization that Flutter has. Python also requires third party support and struggles running on mobile apps due to its interpreted nature, so it must be transformed into appropriate compiled code.

Flutter, and Dart which it is programmed in, have many features to make development faster, easier, and overall less troublesome. One key feature while developing is the ability to "hot reload" the applications to see how changes made to the code alter the application in nearly real time. There is no need to recompile or relaunch the application, you can simply reload with added features and observe the changes made. This is a very useful feature that is possible

through Dart's separation of development and production toolchains. While operating in the development toolchain the language is interpreted and runs slower but doesn't require compilation making it useful during development. This also enables hot reloading by reinterpreting the code and displaying the changes made in an emulating display. Once the app is ready to be distributed, the Dart code can be compiled into ARM code, using the production toolchain, to run on the mobile platform at a much faster speed and less startup time as it can immediately begin executing machine instructions. This duality in toolchains makes it an incredibly powerful language to develop in and maintain all the benefits of both compiled and interpreted languages. This is similar to Java in the sense that the Java Virtual Machine can either use just in time compilation of Java bytecode to interpret the language, or actually compile parts into machine code and save the bytecode interpretation step. However Java cannot run on any mobile devices so it must be compiled before being ported to a mobile device. OCaml and Python must also be compiled into mobile supported languages, with Python having the more difficult task of converting a dynamically typed language into a compiled version.

Dart has many language features which would be very useful in the development of this app. It uses a C like syntax which is easily adaptable for many new users unlike many other mainstream mobile development languages that tend to add unfamiliar new syntax. On top of this Dart utilizes simple but powerful language additions to make such as cascades to limit repeating variable names, mixins to add small additions of functionality to multiple classes, and extension methods to improve existing classes without having to create a subclass. Dart also introduces the idea of a "collection if" to easily implement different functionality on separate platforms. These are novel additions to the language absent from Python, Java, and OCaml and are indicative of the flexibility of Dart.

With a solid UI designed the next step would be to implement the networking capabilities to receive data about how the deals at the garage sale compare to those that can be found online. This will require the use of Dart streams to produce the I/O between the server and app. This is also done through an asynchronous event driven method. Dart uses futures with async and await keywords like asyncio in python to produce concurrent code and improve performance by using non-blocking code. Instead of blocking while waiting for a read, write or connection, the same thread will switch to the next event in its event loop and continue execution. Unlike Python however, Dart allows for multiple threads to run in the same process with the condition that threads do not share memory. These threads are called isolates which are allowed to communicate with each other, but not access each other's memory. This is great for performance because it allows for parallelization if the separation of memory is possible and also enables garbage collection to run on threads that aren't currently running without the threat of altering another thread's memory by mistake. This takes the best parts of both Java and Python by allowing true parallelization of threads like Java, but none of the worries of data races impacting accuracy like Python with its global interpreter lock. OCaml also allows for multithreading but not parallelization like python.

When comparing reliability between languages it is useful to look at the language system used in its implementation including type checking and binding times. Dart is a statically typed, type inference language like OCaml, and Java is also statically typed but with defined types. Python meanwhile is dynamically typed meaning its variables types are unknown until runtime. Inherently statically typed languages are safer and more reliable than dynamically typed

because the compiler can check for type errors while the dynamic language's interpreter will have to deal with the type error when it is thrown. While Dart is statically typed by default, it has the option to explicitly declare a variable to be dynamically typed. This option increases functionality nicely by allowing the option to use a dynamically typed variable while ensuring this action can only be done intentionally, adding an implied level of safety that the user likely understands desired effect and potential consequences of use.

While Java and Python may be more general programming languages with wider library support for a greater array of functionality, Dart is very well optimized for this particular application. Flutter's approach for UI design, Dart's support for parallelizable, asynchronous network communication and also tflite plugin to compute necessary neural network data from the picture taken of the garage sale all provide solid solutions to the key problems in development of GarageGarner. Flutter's system for representing everything as a widget allows for broad customization, however the app will be designed, and Dart's separate development and production toolkits allow for incredible ease of use and eliminate many potential design problems caused by porting Python, Java, or OCaml code into a mobile application. Furthermore Dart is a modern programming language designed in the last decade while all others listed are more that 20 years old. While languages can evolve it's difficult to add revolutionary changes without disrupting existing features. Dart was developed with years of past experience in mind to create a simple and powerful language with great ease of use. On top of that, Flutter which is created and supported by Google, one of the top tech companies in the world, provides an excellent and advanced method for producing the actual app itself. With all this in mind the combination of Dart, Flutter, and tflite is the obvious candidate for developing GarageGarner.

References:
https://flutter.dev/
https://www.tensorflow.org/
https://dart.dev/